

¹

Natural Language Processing

²

Jacob Eisenstein

³

October 5, 2018

4 Contents

| | | |
|-----------|--|-----------|
| 5 | Contents | 1 |
| 6 | Preface | i |
| 7 | Background | i |
| 8 | How to use this book | ii |
| 9 | 1 Introduction | 1 |
| 10 | 1.1 Natural language processing and its neighbors | 1 |
| 11 | 1.2 Three themes in natural language processing | 6 |
| 12 | 1.2.1 Learning and knowledge | 6 |
| 13 | 1.2.2 Search and learning | 7 |
| 14 | 1.2.3 Relational, compositional, and distributional perspectives | 9 |
| 15 | I Learning | 11 |
| 16 | 2 Linear text classification | 13 |
| 17 | 2.1 The bag of words | 13 |
| 18 | 2.2 Naïve Bayes | 16 |
| 19 | 2.2.1 Types and tokens | 19 |
| 20 | 2.2.2 Prediction | 20 |
| 21 | 2.2.3 Estimation | 20 |
| 22 | 2.2.4 Smoothing | 22 |
| 23 | 2.2.5 Setting hyperparameters | 23 |
| 24 | 2.3 Discriminative learning | 24 |
| 25 | 2.3.1 Perceptron | 25 |
| 26 | 2.3.2 Averaged perceptron | 27 |
| 27 | 2.4 Loss functions and large-margin classification | 28 |
| 28 | 2.4.1 Large margin classification | 30 |
| 29 | 2.4.2 Support vector machines | 31 |
| 30 | 2.4.3 Slack variables | 33 |

| | | | |
|----|----------|--|-----------|
| 31 | 2.5 | Logistic regression | 34 |
| 32 | 2.5.1 | Regularization | 35 |
| 33 | 2.5.2 | Gradients | 36 |
| 34 | 2.6 | Optimization | 37 |
| 35 | 2.6.1 | Batch optimization | 37 |
| 36 | 2.6.2 | Online optimization | 38 |
| 37 | 2.7 | *Additional topics in classification | 40 |
| 38 | 2.7.1 | Feature selection by regularization | 40 |
| 39 | 2.7.2 | Other views of logistic regression | 41 |
| 40 | 2.8 | Summary of learning algorithms | 42 |
| 41 | 3 | Nonlinear classification | 47 |
| 42 | 3.1 | Feedforward neural networks | 48 |
| 43 | 3.2 | Designing neural networks | 50 |
| 44 | 3.2.1 | Activation functions | 50 |
| 45 | 3.2.2 | Network structure | 51 |
| 46 | 3.2.3 | Outputs and loss functions | 52 |
| 47 | 3.2.4 | Inputs and lookup layers | 53 |
| 48 | 3.3 | Learning neural networks | 53 |
| 49 | 3.3.1 | Backpropagation | 55 |
| 50 | 3.3.2 | Regularization and dropout | 57 |
| 51 | 3.3.3 | *Learning theory | 58 |
| 52 | 3.3.4 | Tricks | 59 |
| 53 | 3.4 | Convolutional neural networks | 61 |
| 54 | 4 | Linguistic applications of classification | 69 |
| 55 | 4.1 | Sentiment and opinion analysis | 69 |
| 56 | 4.1.1 | Related problems | 71 |
| 57 | 4.1.2 | Alternative approaches to sentiment analysis | 72 |
| 58 | 4.2 | Word sense disambiguation | 73 |
| 59 | 4.2.1 | How many word senses? | 74 |
| 60 | 4.2.2 | Word sense disambiguation as classification | 75 |
| 61 | 4.3 | Design decisions for text classification | 76 |
| 62 | 4.3.1 | What is a word? | 76 |
| 63 | 4.3.2 | How many words? | 79 |
| 64 | 4.3.3 | Count or binary? | 80 |
| 65 | 4.4 | Evaluating classifiers | 81 |
| 66 | 4.4.1 | Precision, recall, and <i>F</i> -MEASURE | 81 |
| 67 | 4.4.2 | Threshold-free metrics | 83 |
| 68 | 4.4.3 | Classifier comparison and statistical significance | 83 |
| 69 | 4.4.4 | *Multiple comparisons | 87 |

| | | |
|-----|---|------------|
| 70 | 4.5 Building datasets | 88 |
| 71 | 4.5.1 Metadata as labels | 88 |
| 72 | 4.5.2 Labeling data | 88 |
| 73 | 5 Learning without supervision | 95 |
| 74 | 5.1 Unsupervised learning | 95 |
| 75 | 5.1.1 K -means clustering | 96 |
| 76 | 5.1.2 Expectation-Maximization (EM) | 98 |
| 77 | 5.1.3 EM as an optimization algorithm | 102 |
| 78 | 5.1.4 How many clusters? | 103 |
| 79 | 5.2 Applications of expectation-maximization | 104 |
| 80 | 5.2.1 Word sense induction | 104 |
| 81 | 5.2.2 Semi-supervised learning | 105 |
| 82 | 5.2.3 Multi-component modeling | 106 |
| 83 | 5.3 Semi-supervised learning | 107 |
| 84 | 5.3.1 Multi-view learning | 108 |
| 85 | 5.3.2 Graph-based algorithms | 109 |
| 86 | 5.4 Domain adaptation | 110 |
| 87 | 5.4.1 Supervised domain adaptation | 111 |
| 88 | 5.4.2 Unsupervised domain adaptation | 112 |
| 89 | 5.5 *Other approaches to learning with latent variables | 114 |
| 90 | 5.5.1 Sampling | 115 |
| 91 | 5.5.2 Spectral learning | 117 |
| 92 | II Sequences and trees | 123 |
| 93 | 6 Language models | 125 |
| 94 | 6.1 N -gram language models | 126 |
| 95 | 6.2 Smoothing and discounting | 129 |
| 96 | 6.2.1 Smoothing | 129 |
| 97 | 6.2.2 Discounting and backoff | 130 |
| 98 | 6.2.3 *Interpolation | 131 |
| 99 | 6.2.4 *Kneser-Ney smoothing | 133 |
| 100 | 6.3 Recurrent neural network language models | 134 |
| 101 | 6.3.1 Backpropagation through time | 136 |
| 102 | 6.3.2 Hyperparameters | 137 |
| 103 | 6.3.3 Gated recurrent neural networks | 137 |
| 104 | 6.4 Evaluating language models | 139 |
| 105 | 6.4.1 Held-out likelihood | 139 |
| 106 | 6.4.2 Perplexity | 140 |

| | | |
|-----|---|------------|
| 107 | 6.5 Out-of-vocabulary words | 141 |
| 108 | 7 Sequence labeling | 145 |
| 109 | 7.1 Sequence labeling as classification | 145 |
| 110 | 7.2 Sequence labeling as structure prediction | 147 |
| 111 | 7.3 The Viterbi algorithm | 149 |
| 112 | 7.3.1 Example | 152 |
| 113 | 7.3.2 Higher-order features | 153 |
| 114 | 7.4 Hidden Markov Models | 153 |
| 115 | 7.4.1 Estimation | 155 |
| 116 | 7.4.2 Inference | 155 |
| 117 | 7.5 Discriminative sequence labeling with features | 157 |
| 118 | 7.5.1 Structured perceptron | 160 |
| 119 | 7.5.2 Structured support vector machines | 160 |
| 120 | 7.5.3 Conditional random fields | 162 |
| 121 | 7.6 Neural sequence labeling | 167 |
| 122 | 7.6.1 Recurrent neural networks | 167 |
| 123 | 7.6.2 Character-level models | 169 |
| 124 | 7.6.3 Convolutional Neural Networks for Sequence Labeling | 170 |
| 125 | 7.7 *Unsupervised sequence labeling | 170 |
| 126 | 7.7.1 Linear dynamical systems | 172 |
| 127 | 7.7.2 Alternative unsupervised learning methods | 172 |
| 128 | 7.7.3 Semiring notation and the generalized viterbi algorithm | 172 |
| 129 | 8 Applications of sequence labeling | 175 |
| 130 | 8.1 Part-of-speech tagging | 175 |
| 131 | 8.1.1 Parts-of-Speech | 176 |
| 132 | 8.1.2 Accurate part-of-speech tagging | 180 |
| 133 | 8.2 Morphosyntactic Attributes | 182 |
| 134 | 8.3 Named Entity Recognition | 183 |
| 135 | 8.4 Tokenization | 185 |
| 136 | 8.5 Code switching | 186 |
| 137 | 8.6 Dialogue acts | 187 |
| 138 | 9 Formal language theory | 191 |
| 139 | 9.1 Regular languages | 192 |
| 140 | 9.1.1 Finite state acceptors | 193 |
| 141 | 9.1.2 Morphology as a regular language | 194 |
| 142 | 9.1.3 Weighted finite state acceptors | 196 |
| 143 | 9.1.4 Finite state transducers | 201 |
| 144 | 9.1.5 *Learning weighted finite state automata | 206 |

| | | |
|-----|--|------------|
| 145 | 9.2 Context-free languages | 207 |
| 146 | 9.2.1 Context-free grammars | 208 |
| 147 | 9.2.2 Natural language syntax as a context-free language | 211 |
| 148 | 9.2.3 A phrase-structure grammar for English | 213 |
| 149 | 9.2.4 Grammatical ambiguity | 218 |
| 150 | 9.3 *Mildly context-sensitive languages | 218 |
| 151 | 9.3.1 Context-sensitive phenomena in natural language | 219 |
| 152 | 9.3.2 Combinatory categorial grammar | 220 |
| 153 | 10 Context-free parsing | 225 |
| 154 | 10.1 Deterministic bottom-up parsing | 226 |
| 155 | 10.1.1 Recovering the parse tree | 227 |
| 156 | 10.1.2 Non-binary productions | 227 |
| 157 | 10.1.3 Complexity | 229 |
| 158 | 10.2 Ambiguity | 229 |
| 159 | 10.2.1 Parser evaluation | 230 |
| 160 | 10.2.2 Local solutions | 231 |
| 161 | 10.3 Weighted Context-Free Grammars | 232 |
| 162 | 10.3.1 Parsing with weighted context-free grammars | 234 |
| 163 | 10.3.2 Probabilistic context-free grammars | 235 |
| 164 | 10.3.3 *Semiring weighted context-free grammars | 237 |
| 165 | 10.4 Learning weighted context-free grammars | 238 |
| 166 | 10.4.1 Probabilistic context-free grammars | 238 |
| 167 | 10.4.2 Feature-based parsing | 239 |
| 168 | 10.4.3 *Conditional random field parsing | 240 |
| 169 | 10.4.4 Neural context-free grammars | 242 |
| 170 | 10.5 Grammar refinement | 242 |
| 171 | 10.5.1 Parent annotations and other tree transformations | 243 |
| 172 | 10.5.2 Lexicalized context-free grammars | 244 |
| 173 | 10.5.3 *Refinement grammars | 248 |
| 174 | 10.6 Beyond context-free parsing | 249 |
| 175 | 10.6.1 Reranking | 250 |
| 176 | 10.6.2 Transition-based parsing | 251 |
| 177 | 11 Dependency parsing | 257 |
| 178 | 11.1 Dependency grammar | 257 |
| 179 | 11.1.1 Heads and dependents | 258 |
| 180 | 11.1.2 Labeled dependencies | 259 |
| 181 | 11.1.3 Dependency subtrees and constituents | 260 |
| 182 | 11.2 Graph-based dependency parsing | 262 |
| 183 | 11.2.1 Graph-based parsing algorithms | 264 |

| | | |
|-----|---|------------|
| 184 | 11.2.2 Computing scores for dependency arcs | 265 |
| 185 | 11.2.3 Learning | 267 |
| 186 | 11.3 Transition-based dependency parsing | 268 |
| 187 | 11.3.1 Transition systems for dependency parsing | 269 |
| 188 | 11.3.2 Scoring functions for transition-based parsers | 273 |
| 189 | 11.3.3 Learning to parse | 274 |
| 190 | 11.4 Applications | 277 |
| 191 | III Meaning | 283 |
| 192 | 12 Logical semantics | 285 |
| 193 | 12.1 Meaning and denotation | 286 |
| 194 | 12.2 Logical representations of meaning | 287 |
| 195 | 12.2.1 Propositional logic | 287 |
| 196 | 12.2.2 First-order logic | 288 |
| 197 | 12.3 Semantic parsing and the lambda calculus | 291 |
| 198 | 12.3.1 The lambda calculus | 292 |
| 199 | 12.3.2 Quantification | 293 |
| 200 | 12.4 Learning semantic parsers | 296 |
| 201 | 12.4.1 Learning from derivations | 297 |
| 202 | 12.4.2 Learning from logical forms | 299 |
| 203 | 12.4.3 Learning from denotations | 301 |
| 204 | 13 Predicate-argument semantics | 305 |
| 205 | 13.1 Semantic roles | 307 |
| 206 | 13.1.1 VerbNet | 308 |
| 207 | 13.1.2 Proto-roles and PropBank | 309 |
| 208 | 13.1.3 FrameNet | 310 |
| 209 | 13.2 Semantic role labeling | 312 |
| 210 | 13.2.1 Semantic role labeling as classification | 312 |
| 211 | 13.2.2 Semantic role labeling as constrained optimization | 315 |
| 212 | 13.2.3 Neural semantic role labeling | 317 |
| 213 | 13.3 Abstract Meaning Representation | 318 |
| 214 | 13.3.1 AMR Parsing | 321 |
| 215 | 14 Distributional and distributed semantics | 325 |
| 216 | 14.1 The distributional hypothesis | 325 |
| 217 | 14.2 Design decisions for word representations | 327 |
| 218 | 14.2.1 Representation | 327 |
| 219 | 14.2.2 Context | 328 |

| | | |
|-----|---|------------|
| 220 | 14.2.3 Estimation | 329 |
| 221 | 14.3 Latent semantic analysis | 329 |
| 222 | 14.4 Brown clusters | 331 |
| 223 | 14.5 Neural word embeddings | 334 |
| 224 | 14.5.1 Continuous bag-of-words (CBOW) | 334 |
| 225 | 14.5.2 Skipgrams | 335 |
| 226 | 14.5.3 Computational complexity | 335 |
| 227 | 14.5.4 Word embeddings as matrix factorization | 337 |
| 228 | 14.6 Evaluating word embeddings | 338 |
| 229 | 14.6.1 Intrinsic evaluations | 339 |
| 230 | 14.6.2 Extrinsic evaluations | 339 |
| 231 | 14.6.3 Fairness and bias | 340 |
| 232 | 14.7 Distributed representations beyond distributional statistics | 341 |
| 233 | 14.7.1 Word-internal structure | 341 |
| 234 | 14.7.2 Lexical semantic resources | 343 |
| 235 | 14.8 Distributed representations of multiword units | 344 |
| 236 | 14.8.1 Purely distributional methods | 344 |
| 237 | 14.8.2 Distributional-compositional hybrids | 345 |
| 238 | 14.8.3 Supervised compositional methods | 346 |
| 239 | 14.8.4 Hybrid distributed-symbolic representations | 346 |
| 240 | 15 Reference Resolution | 351 |
| 241 | 15.1 Forms of referring expressions | 352 |
| 242 | 15.1.1 Pronouns | 352 |
| 243 | 15.1.2 Proper Nouns | 357 |
| 244 | 15.1.3 Nominals | 358 |
| 245 | 15.2 Algorithms for coreference resolution | 358 |
| 246 | 15.2.1 Mention-pair models | 359 |
| 247 | 15.2.2 Mention-ranking models | 360 |
| 248 | 15.2.3 Transitive closure in mention-based models | 361 |
| 249 | 15.2.4 Entity-based models | 362 |
| 250 | 15.3 Representations for coreference resolution | 367 |
| 251 | 15.3.1 Features | 368 |
| 252 | 15.3.2 Distributed representations of mentions and entities | 370 |
| 253 | 15.4 Evaluating coreference resolution | 373 |
| 254 | 16 Discourse | 379 |
| 255 | 16.1 Segments | 379 |
| 256 | 16.1.1 Topic segmentation | 380 |
| 257 | 16.1.2 Functional segmentation | 381 |
| 258 | 16.2 Entities and reference | 381 |

| | | |
|-----|---|------------|
| 259 | 16.2.1 Centering theory | 382 |
| 260 | 16.2.2 The entity grid | 383 |
| 261 | 16.2.3 *Formal semantics beyond the sentence level | 384 |
| 262 | 16.3 Relations | 385 |
| 263 | 16.3.1 Shallow discourse relations | 385 |
| 264 | 16.3.2 Hierarchical discourse relations | 389 |
| 265 | 16.3.3 Argumentation | 392 |
| 266 | 16.3.4 Applications of discourse relations | 393 |
| 267 | IV Applications | 401 |
| 268 | 17 Information extraction | 403 |
| 269 | 17.1 Entities | 405 |
| 270 | 17.1.1 Entity linking by learning to rank | 406 |
| 271 | 17.1.2 Collective entity linking | 408 |
| 272 | 17.1.3 *Pairwise ranking loss functions | 409 |
| 273 | 17.2 Relations | 411 |
| 274 | 17.2.1 Pattern-based relation extraction | 412 |
| 275 | 17.2.2 Relation extraction as a classification task | 413 |
| 276 | 17.2.3 Knowledge base population | 416 |
| 277 | 17.2.4 Open information extraction | 419 |
| 278 | 17.3 Events | 420 |
| 279 | 17.4 Hedges, denials, and hypotheticals | 422 |
| 280 | 17.5 Question answering and machine reading | 424 |
| 281 | 17.5.1 Formal semantics | 424 |
| 282 | 17.5.2 Machine reading | 425 |
| 283 | 18 Machine translation | 431 |
| 284 | 18.1 Machine translation as a task | 431 |
| 285 | 18.1.1 Evaluating translations | 433 |
| 286 | 18.1.2 Data | 435 |
| 287 | 18.2 Statistical machine translation | 436 |
| 288 | 18.2.1 Statistical translation modeling | 437 |
| 289 | 18.2.2 Estimation | 438 |
| 290 | 18.2.3 Phrase-based translation | 439 |
| 291 | 18.2.4 *Syntax-based translation | 441 |
| 292 | 18.3 Neural machine translation | 442 |
| 293 | 18.3.1 Neural attention | 444 |
| 294 | 18.3.2 *Neural machine translation without recurrence | 446 |
| 295 | 18.3.3 Out-of-vocabulary words | 447 |

| | | |
|-----|---|------------|
| 296 | 18.4 Decoding | 449 |
| 297 | 18.5 Training towards the evaluation metric | 451 |
| 298 | 19 Text generation | 457 |
| 299 | 19.1 Data-to-text generation | 457 |
| 300 | 19.1.1 Latent data-to-text alignment | 459 |
| 301 | 19.1.2 Neural data-to-text generation | 460 |
| 302 | 19.2 Text-to-text generation | 464 |
| 303 | 19.2.1 Neural abstractive summarization | 464 |
| 304 | 19.2.2 Sentence fusion for multi-document summarization | 465 |
| 305 | 19.3 Dialogue | 466 |
| 306 | 19.3.1 Finite-state and agenda-based dialogue systems | 467 |
| 307 | 19.3.2 Markov decision processes | 468 |
| 308 | 19.3.3 Neural chatbots | 470 |
| 309 | A Probability | 475 |
| 310 | A.1 Probabilities of event combinations | 475 |
| 311 | A.1.1 Probabilities of disjoint events | 476 |
| 312 | A.1.2 Law of total probability | 477 |
| 313 | A.2 Conditional probability and Bayes' rule | 477 |
| 314 | A.3 Independence | 479 |
| 315 | A.4 Random variables | 480 |
| 316 | A.5 Expectations | 481 |
| 317 | A.6 Modeling and estimation | 482 |
| 318 | B Numerical optimization | 485 |
| 319 | B.1 Gradient descent | 486 |
| 320 | B.2 Constrained optimization | 486 |
| 321 | B.3 Example: Passive-aggressive online learning | 487 |
| 322 | Bibliography | 489 |

³²³ Preface

³²⁴ This text began with notes that I wrote for Georgia Tech’s undergraduate and gradu-
³²⁵ ate courses on natural language processing, CS 4650 and 7650. There are several other
³²⁶ good resources (e.g., Manning and Schütze, 1999; Jurafsky and Martin, 2009; Smith, 2011;
³²⁷ Collins, 2013), but the goal of this text is focus on a core subset of the field, unified by the
³²⁸ concepts of learning and search. A remarkable thing about natural language processing
³²⁹ is that so many problems can be solved by a compact set of methods:

³³⁰ **Search.** Viterbi, CKY, minimum spanning tree, shift-reduce, integer linear programming,
³³¹ beam search.

³³² **Learning.** Maximum-likelihood estimation, logistic regression, perceptron, expectation-
³³³ maximization, matrix factorization, backpropagation.

³³⁴ This text explains how these methods work, and how they can be applied to problems
³³⁵ that arise in the computer processing of natural language: document classification, word
³³⁶ sense disambiguation, sequence labeling (part-of-speech tagging and named entity recog-
³³⁷ nition), parsing, coreference resolution, relation extraction, discourse analysis, language
³³⁸ modeling, and machine translation.

³³⁹ Background

³⁴⁰ Because natural language processing draws on many different intellectual traditions, al-
³⁴¹ most everyone who approaches it feels underprepared in one way or another. Here is a
³⁴² summary of what is expected, and where you can learn more:

³⁴³ **Mathematics and machine learning.** The text assumes a background in multivariate cal-
³⁴⁴ culus and linear algebra: vectors, matrices, derivatives, and partial derivatives. You
³⁴⁵ should also be familiar with probability and statistics. A review of basic proba-
³⁴⁶ bility is found in Appendix A, and a minimal review of numerical optimization is
³⁴⁷ found in Appendix B. For linear algebra, the online course and textbook from Strang
³⁴⁸ (2016) are excellent sources of review material. Deisenroth et al. (2018) are currently

349 preparing a textbook on *Mathematics for Machine Learning*, and several chapters can
350 be found online.¹ For an introduction to probabilistic modeling and estimation, see
351 James et al. (2013); for a more advanced and comprehensive discussion of the same
352 material, the classic reference is Hastie et al. (2009).

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

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

368 How to use this book

369 After the introduction, the rest of the textbook is organized into four main units:

370 **Learning.** This section builds up a set of machine learning tools that will be used through-
371 out the other sections. Because the focus is on machine learning, the text repres-
372 entations and linguistic phenomena are mostly simple: “bag-of-words” text classifi-
373 cation is treated as a model example. Chapter 4 describes some of the more linguisti-
374 cally interesting applications of word-based text analysis.

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

380 **Meaning.** This section takes a broad view of efforts to represent and compute meaning
381 from text, ranging from formal logic to neural word embeddings. It also includes

¹<https://mml-book.github.io/>

382 two topics that are closely related to semantics: resolution of ambiguous references,
 383 and analysis of multi-sentence discourse structure.

384 **Applications.** The final section offers chapter-length treatments on three of the most promi-
 385 nent applications of natural language processing: information extraction, machine
 386 translation, and text generation. Each of these applications merits a textbook length
 387 treatment of its own (Koehn, 2009; Grishman, 2012; Reiter and Dale, 2000); the chap-
 388 ters here explain some of the most well known systems using the formalisms and
 389 methods built up earlier in the book, while introducing methods such as neural at-
 390 tention.

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

395 Chapters 1-3 provide building blocks that will be used throughout the book, and chap-
 396 ter 4 describes some critical aspects of the practice of language technology. Language
 397 models (chapter 6), sequence labeling (chapter 7), and parsing (chapter 10 and 11) are
 398 canonical topics in natural language processing, and distributed word embeddings (chap-
 399 ter 14) have become ubiquitous. Of the applications, machine translation (chapter 18) is
 400 the best choice: it is more cohesive than information extraction, and more mature than text
 401 generation. In my experience, nearly all students benefit from the review of probability in
 402 Appendix A.

- 403 • A course focusing on machine learning should add the chapter on unsupervised
 404 learning (chapter 5). The chapters on predicate-argument semantics (chapter 13),
 405 reference resolution (chapter 15), and text generation (chapter 19) are particularly
 406 influenced by recent machine learning innovations, including deep neural networks
 407 and learning to search.
- 408 • A course with a more linguistic orientation should add the chapters on applica-
 409 tions of sequence labeling (chapter 8), formal language theory (chapter 9), semantics
 410 (chapter 12 and 13), and discourse (chapter 16).
- 411 • For a course with a more applied focus, I recommend the chapters on applications
 412 of sequence labeling (chapter 8), predicate-argument semantics (chapter 13), infor-
 413 mation extraction (chapter 17), and text generation (chapter 19).

414 Acknowledgments

415 Several colleagues, students, and friends read early drafts of chapters in their areas of
 416 expertise, including Yoav Artzi, Kevin Duh, Heng Ji, Jessy Li, Brendan O'Connor, Yuval

417 Pinter, Shawn Ling Ramirez, Nathan Schneider, Pamela Shapiro, Noah A. Smith, Sandeep
418 Soni, and Luke Zettlemoyer. I also thank the anonymous reviewers, particularly reviewer
419 4, who provided detailed line-by-line edits and suggestions. The text benefited from dis-
420 cussions with my editor Marie Lufkin Lee, as well as Kevin Murphy, Shawn Ling Ramirez,
421 and Bonnie Webber. In addition, there are many students, colleagues, friends, and family
422 who found mistakes in early drafts, or who recommended key references. These include:
423 Parminder Bhatia, Kimberly Caras, Jiahao Cai, Justin Chen, Murtaza Dhuliawala, Yantao
424 Du, Barbara Eisenstein, Luiz C. F. Ribeiro, Chris Gu, Joshua Killingsworth, Jonathan May,
425 Taha Merghani, Gus Monod, Raghavendra Murali, Nidish Nair, Brendan O'Connor, Bran-
426 don Peck, Yuval Pinter, Nathan Schneider, Jianhao Shen, Zhewei Sun, Rubin Tsui, Ashwin
427 Cunnappakkam Vinjimir, Denny Vrandečić, William Yang Wang, Clay Washington, Ishan
428 Waykul, Xavier Yao, Yuyu Zhang, and also some anonymous commenters. Clay Wash-
429 ington tested several of the programming exercises. Special thanks to the many students
430 in Georgia Tech's CS 4650 and 7650 who suffered through early versions of the text.

431 Notation

432 As a general rule, words, word counts, and other types of observations are indicated with
433 Roman letters (a, b, c); parameters are indicated with Greek letters (α, β, θ). Vectors are
434 indicated with bold script for both random variables x and parameters θ . Other useful
435 notations are indicated in the table below.

Basics

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

Linear algebra

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

Text datasets

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

Probabilities

| | |
|------------------|--|
| $\Pr(A)$ | probability of event A |
| $\Pr(A B)$ | probability of event A , conditioned on event B |
| $p_B(b)$ | the marginal probability of random variable B taking value b ; written $p(b)$ when the choice of random variable is clear from context |
| $p_{B A}(b a)$ | the probability of random variable B taking value b , conditioned on A taking value a ; written $p(b a)$ when clear from context |
| $A \sim p$ | the random variable A is distributed according to distribution p . For example, $X \sim \mathcal{N}(0, 1)$ states that the random variable X is drawn from a normal distribution with zero mean and unit variance. |
| $A B \sim p$ | conditioned on the random variable B , A is distributed according to p . ² |

Machine learning

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

⁴³⁶ **Chapter 1**

⁴³⁷ **Introduction**

⁴³⁸ Natural language processing is the set of methods for making human language accessible
⁴³⁹ to computers. In the past decade, natural language processing has become embedded
⁴⁴⁰ in our daily lives: automatic machine translation is ubiquitous on the web and in social
⁴⁴¹ media; text classification keeps emails from collapsing under a deluge of spam; search
⁴⁴² engines have moved beyond string matching and network analysis to a high degree of
⁴⁴³ linguistic sophistication; dialog systems provide an increasingly common and effective
⁴⁴⁴ way to get and share information.

⁴⁴⁵ These diverse applications are based on a common set of ideas, drawing on algo-
⁴⁴⁶ rithms, linguistics, logic, statistics, and more. The goal of this text is to provide a survey
⁴⁴⁷ of these foundations. The technical fun starts in the next chapter; the rest of this current
⁴⁴⁸ chapter situates natural language processing with respect to other intellectual disciplines,
⁴⁴⁹ identifies some high-level themes in contemporary natural language processing, and ad-
⁴⁵⁰ vises the reader on how best to approach the subject.

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

⁴⁵² Natural language processing draws on many other intellectual traditions, from formal
⁴⁵³ linguistics to statistical physics. This section briefly situates natural language processing
⁴⁵⁴ with respect to some of its closest neighbors.

⁴⁵⁵ **Computational Linguistics** Most of the meetings and journals that host natural lan-
⁴⁵⁶ guage processing research bear the name “computational linguistics”, and the terms may
⁴⁵⁷ be thought of as essentially synonymous. But while there is substantial overlap, there is
⁴⁵⁸ an important difference in focus. In linguistics, language is the object of study. Compu-
⁴⁵⁹ tational methods may be brought to bear, just as in scientific disciplines like computational
⁴⁶⁰ biology and computational astronomy, but they play only a supporting role. In contrast,

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

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

- 476 • Unlike images or audio, text data is fundamentally discrete, with meaning created
 477 by combinatorial arrangements of symbolic units. This is particularly consequential
 478 for applications in which text is the output, such as translation and summarization,
 479 because it is not possible to gradually approach an optimal solution.
 - 480 • Although the set of words is discrete, new words are always being created. Further-
 481 more, the distribution over words (and other linguistic elements) resembles that of a
 482 **power law**¹ (Zipf, 1949): there will be a few words that are very frequent, and a long
 483 tail of words that are rare. A consequence is that natural language processing algo-
 484 rithms must be especially robust to observations that do not occur in the training
 485 data.
 - 486 • Language is **compositional**: units such as words can combine to create phrases,
 487 which can combine by the very same principles to create larger phrases. For ex-
 488 ample, a **noun phrase** can be created by combining a smaller noun phrase with a
 489 **prepositional phrase**, as in *the whiteness of the whale*. The prepositional phrase is
 490 created by combining a preposition (in this case, *of*) with another noun phrase (*the
 491 whale*). In this way, it is possible to create arbitrarily long phrases, such as,
- 492 (1.1) ...huge globular pieces of the whale of the bigness of a human head.²

493 The meaning of such a phrase must be analyzed in accord with the underlying hier-
 494 archical structure. In this case, *huge globular pieces of the whale* acts as a single noun

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

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

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

500 **Artificial Intelligence** The goal of artificial intelligence is to build software and robots
501 with the same range of abilities as humans (Russell and Norvig, 2009). Natural language
502 processing is relevant to this goal in several ways. On the most basic level, the capacity for
503 language is one of the central features of human intelligence, and is therefore a prerequi-
504 site for artificial intelligence.³ Second, much of artificial intelligence research is dedicated
505 to the development of systems that can reason from premises to a conclusion, but such
506 algorithms are only as good as what they know (Dreyfus, 1992). Natural language pro-
507 cessing is a potential solution to the “knowledge bottleneck”, by acquiring knowledge
508 from texts, and perhaps also from conversations. This idea goes all the way back to Tur-
509 ing’s 1949 paper *Computing Machinery and Intelligence*, which proposed the **Turing test** for
510 determining whether artificial intelligence had been achieved (Turing, 2009).

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

515 (1.2) The trophy doesn’t fit into the brown suitcase because **it** is too [small/large].
516 When the final word is *small*, then the pronoun *it* refers to the suitcase; when the final
517 word is *large*, then *it* refers to the trophy. Solving this example requires spatial reasoning;
518 other schemas require reasoning about actions and their effects, emotions and intentions,
519 and social conventions.

520 Such examples demonstrate that natural language understanding cannot be achieved
521 in isolation from knowledge and reasoning. Yet the history of artificial intelligence has
522 been one of increasing specialization: with the growing volume of research in subdisci-
523 plines such as natural language processing, machine learning, and computer vision, it is

³This view is shared by some, but not all, prominent researchers in artificial intelligence. Michael Jordan, a specialist in machine learning, has said that if he had a billion dollars to spend on any large research project, he would spend it on natural language processing (https://www.reddit.com/r/MachineLearning/comments/2fxi6v/ama_michael_i_jordan/). On the other hand, in a public discussion about the future of artificial intelligence in February 2018, computer vision researcher Yann Lecun argued that despite its many practical applications, language is perhaps “number 300” in the priority list for artificial intelligence research, and that it would be a great achievement if AI could attain the capabilities of an orangutan, which do not include language (<http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html>).

524 difficult for anyone to maintain expertise across the entire field. Still, recent work has
525 demonstrated interesting connections between natural language processing and other ar-
526 eas of AI, including computer vision (e.g., Antol et al., 2015) and game playing (e.g.,
527 Branavan et al., 2009). The dominance of machine learning throughout artificial intel-
528 ligence has led to a broad consensus on representations such as graphical models and
529 computation graphs, and on algorithms such as backpropagation and combinatorial opti-
530 mization. Many of the algorithms and representations covered in this text are part of this
531 consensus.

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

540 The study of computer systems is also relevant to natural language processing. Large
541 datasets of unlabeled text can be processed more quickly by parallelization techniques
542 like MapReduce (Dean and Ghemawat, 2008; Lin and Dyer, 2010); high-volume data
543 sources such as social media can be summarized efficiently by approximate streaming
544 and sketching techniques (Goyal et al., 2009). When deep neural networks are imple-
545 mented in production systems, it is possible to eke out speed gains using techniques such
546 as reduced-precision arithmetic (Wu et al., 2016). Many classical natural language process-
547 ing algorithms are not naturally suited to graphics processing unit (GPU) parallelization,
548 suggesting directions for further research at the intersection of natural language process-
549 ing and computing hardware (Yi et al., 2011).

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

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

563 **Ethics** As machine learning and artificial intelligence become increasingly ubiquitous, it
564 is crucial to understand how their benefits, costs, and risks are distributed across differ-
565 ent kinds of people. Natural language processing raises some particularly salient issues
566 around **ethics, fairness, and accountability**:

567 **Access.** Who is natural language processing designed to serve? For example, whose lan-
568 guage is translated *from*, and whose language is translated *to*?

569 **Bias.** Does language technology learn to replicate social biases from text corpora, and
570 does it reinforce these biases as seemingly objective computational conclusions?

571 **Labor.** Whose text and speech comprise the datasets that power natural language pro-
572 cessing, and who performs the annotations? Are the benefits of this technology
573 shared with all the people whose work makes it possible?

574 **Privacy and internet freedom.** What is the impact of large-scale text processing on the
575 right to free and private communication? What is the potential role of natural lan-
576 guage processing in regimes of censorship or surveillance?

577 This text lightly touches on issues related to fairness and bias in § 14.6.3 and § 18.1.1,
578 but these issues are worthy of a book of their own. For more from within the field of
579 computational linguistics, see the papers from the annual workshop on Ethics in Natural
580 Language Processing (Hovy et al., 2017; Alfano et al., 2018). For an outside perspective on
581 ethical issues relating to data science at large, see boyd and Crawford (2012).

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

593 1.2 Three themes in natural language processing

594 Natural language processing covers a diverse range of tasks, methods, and linguistic phe-
 595 nomena. But despite the apparent incommensurability between, say, the summarization
 596 of scientific articles (§ 16.3.4) and the identification of suffix patterns in Spanish verbs
 597 (§ 9.1.4), some general themes emerge. The remainder of the introduction focuses on these
 598 themes, which will recur in various forms through the text. Each theme can be expressed
 599 as an opposition between two extreme viewpoints on how to process natural language.
 600 The methods discussed in the text can usually be placed somewhere on the continuum
 601 between these two extremes.

602 1.2.1 Learning and knowledge

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

612 The end-to-end approach has been buoyed by recent results in computer vision and
 613 speech recognition, in which advances in machine learning have swept away expert-
 614 engineered representations based on the fundamentals of optics and phonology (Krizhevsky
 615 et al., 2012; Graves and Jaitly, 2014). But while machine learning is an element of nearly
 616 every contemporary approach to natural language processing, linguistic representations
 617 such as syntax trees have not yet gone the way of the visual edge detector or the auditory
 618 triphone. Linguists have argued for the existence of a “language faculty” in all human be-
 619 ings, which encodes a set of abstractions specially designed to facilitate the understanding
 620 and production of language. The argument for the existence of such a language faculty
 621 is based on the observation that children learn language faster and from fewer examples
 622 than would be possible if language was learned from experience alone.⁴ From a practi-
 623 cal standpoint, linguistic structure seems to be particularly important in scenarios where
 624 training data is limited.

625 There are a number of ways in which knowledge and learning can be combined in
 626 natural language processing. Many supervised learning systems make use of carefully
 627 engineered **features**, which transform the data into a representation that can facilitate

⁴The *Language Instinct* (Pinker, 2003) articulates these arguments in an engaging and popular style. For arguments against the innateness of language, see Elman et al. (1998).

learning. For example, in a task like search, it may be useful to identify each word's **stem**, so that a system can more easily generalize across related terms such as *whale*, *whales*, *whalers*, and *whaling*. (This issue is relatively benign in English, as compared to the many other languages which include much more elaborate systems of prefixed and suffixes.) Such features could be obtained from a hand-crafted resource, like a dictionary that maps each word to a single root form. Alternatively, features can be obtained from the output of a general-purpose language processing system, such as a parser or part-of-speech tagger, which may itself be built on supervised machine learning.

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

The debate about the relative importance of machine learning and linguistic knowledge sometimes becomes heated. No machine learning specialist likes to be told that their engineering methodology is unscientific alchemy;⁵ nor does a linguist want to hear that the search for general linguistic principles and structures has been made irrelevant by big data. Yet there is clearly room for both types of research: we need to know how far we can go with end-to-end learning alone, while at the same time, we continue the search for linguistic representations that generalize across applications, scenarios, and languages. For more on the history of this debate, see Church (2011); for an optimistic view of the potential symbiosis between computational linguistics and deep learning, see Manning (2015).

1.2.2 Search and learning

Many natural language processing problems can be written mathematically in the form of optimization,⁶

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

where,

- x is the input, which is an element of a set \mathcal{X} ;
- y is the output, which is an element of a set $\mathcal{Y}(x)$;

⁵Ali Rahimi argued that much of deep learning research was similar to “alchemy” in a presentation at the 2017 conference on Neural Information Processing Systems. He was advocating for more learning theory, not more linguistics.

⁶Throughout this text, equations will be numbered by square brackets, and linguistic examples will be numbered by parentheses.

- Ψ is a scoring function (also called the **model**), which maps from the set $\mathcal{X} \times \mathcal{Y}$ to the real numbers;
- θ is a vector of parameters for Ψ ;
- \hat{y} is the predicted output, which is chosen to maximize the scoring function.

This basic structure can be applied to a huge range of problems. For example, the input x might be a social media post, and the output y might be a labeling of the emotional sentiment expressed by the author (chapter 4); or x could be a sentence in French, and the output y could be a sentence in Tamil (chapter 18); or x might be a sentence in English, and y might be a representation of the syntactic structure of the sentence (chapter 10); or x might be a news article and y might be a structured record of the events that the article describes (chapter 17).

This formulation reflects an implicit decision that language processing algorithms will have two distinct modules:

Search. The search module is responsible for computing the argmax of the function Ψ . In other words, it finds the output \hat{y} that gets the best score with respect to the input x . This is easy when the search space $\mathcal{Y}(x)$ is small enough to enumerate, or when the scoring function Ψ has a convenient decomposition into parts. In many cases, we will want to work with scoring functions that do not have these properties, motivating the use of more sophisticated search algorithms, such as bottom-up dynamic programming (§ 10.1) and beam search (§ 11.3.1). Because the outputs are usually discrete in language processing problems, search often relies on the machinery of **combinatorial optimization**.

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

The division of natural language processing into separate modules for search and learning makes it possible to reuse generic algorithms across many tasks and models. Much of the work of natural language processing can be focused on the design of the model Ψ — identifying and formalizing the linguistic phenomena that are relevant to the task at hand — while reaping the benefits of decades of progress in search, optimization, and learning. This textbook will describe several classes of scoring functions, and the corresponding algorithms for search and learning.

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

1.2.3 Relational, compositional, and distributional perspectives

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

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

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

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

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

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

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

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

- 751 (1.4) a. The blubber served them as fuel.
- 752 b. ...extracting it from the blubber of the large fish ...
- 753 c. Amongst oily substances, blubber has been employed as a manure.

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

762 The relational, compositional, and distributional perspectives all contribute to our un-
 763 derstanding of linguistic meaning, and all three appear to be critical to natural language
 764 processing. Yet they are uneasy collaborators, requiring seemingly incompatible represen-
 765 tations and algorithmic approaches. This text presents some of the best known and most
 766 successful methods for working with each of these representations, but future research
 767 may reveal new ways to combine them.

768

Part I

769

Learning

770 **Chapter 2**

771 **Linear text classification**

772 We begin with the problem of **text classification**: given a text document, assign it a discrete label $y \in \mathcal{Y}$, where \mathcal{Y} is the set of possible labels. This problem has many applications, from spam filtering to analysis of electronic health records. This chapter describes some of the most well known and effective algorithms for text classification, from a mathematical perspective that should help you understand what they do and why they work.
773 774 775 776 777 778 779 780 781 Text classification is also a building block that is used in more elaborate natural language processing tasks. For readers without a background in machine learning or statistics, the material in this chapter will take more time to digest than most of the subsequent chapters. But this investment will pay off as the mathematical principles behind these basic classification algorithms reappear in other contexts throughout the book.

782 **2.1 The bag of words**

783 To perform text classification, the first question is how to represent each document, or instance. A common approach is to use a column vector of word counts, e.g., $\mathbf{x} = [0, 1, 1, 0, 0, 2, 0, 1, 13, 0 \dots]^T$, where 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 in the vocabulary. In linear classification, the classification decision is based on a weighted sum of individual feature counts, such as word counts.
784 785 786 787 788

789 The object \mathbf{x} is a vector, but it is often called a **bag of words**, because it includes only information about the count of each word, and not the order in which the words appear.
790 With the bag of words representation, we are ignoring grammar, sentence boundaries,
791 paragraphs — everything but the words. Yet the bag of words model is surprisingly
792 effective for text classification. If you see the word *whale* in a document, is it fiction or non-
793 fiction? What if you see the word *molybdenum*? For many labeling problems, individual
794 words can be strong predictors.
795

796 To predict a label from a bag-of-words, we can assign a score to each word in the vo-
 797 cabulary, measuring the compatibility with the label. For example, for the label FICTION,
 798 we might assign a positive score to the word *whale*, and a negative score to the word
 799 *modlybdenum*. These scores are called **weights**, and they are arranged in a column vector
 800 θ .

801 Suppose that you want a multiclass classifier, where $K \triangleq |\mathcal{Y}| > 2$. For example, you
 802 might want to classify news stories about sports, celebrities, music, and business. The goal
 803 is to predict a label \hat{y} , given the bag of words x , using the weights θ . For each label $y \in \mathcal{Y}$,
 804 we compute a score $\Psi(x, y)$, which is a scalar measure of the compatibility between the
 805 bag-of-words x and the label y . In a linear bag-of-words classifier, this score is the vector
 806 inner product between the weights θ and the output of a **feature function** $f(x, y)$,

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

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

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

810 This function returns the count of the word *whale* if the label is FICTION, and it returns zero
 811 otherwise. The index j depends on the position of *whale* in the vocabulary, and of FICTION
 812 in the set of possible labels. The corresponding weight θ_j then scores the compatibility of
 813 the word *whale* with the label FICTION.¹ A positive score means that this word makes the
 814 label more likely.

The output of the feature function can be formalized as a vector:

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

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

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

815 where $\underbrace{[0; 0; \dots; 0]}_{(K-1) \times V}$ is a column vector of $(K - 1) \times V$ zeros, and the semicolon indicates
 816 vertical concatenation. For each of the K possible labels, the feature function returns a

¹In practice, both f and θ may be implemented as a dictionary rather than vectors, so that it is not necessary to explicitly identify j . In such an implementation, the tuple (*whale*, FICTION) acts as a key in both dictionaries; the values in f are feature counts, and the values in θ are weights.

817 vector that is mostly zeros, with a column vector of word counts \mathbf{x} inserted in a location
 818 that depends on the specific label y . This arrangement is shown in Figure 2.1; the notation
 819 may seem awkward at first, but it generalizes to an impressive range of learning settings.

Given a vector of weights, $\boldsymbol{\theta} \in \mathbb{R}^{VK}$, we can now compute the score $\Psi(\mathbf{x}, y)$ by Equation 2.1. This inner product gives a scalar measure of the compatibility of the observation \mathbf{x} with label y .² For any document \mathbf{x} , we predict the label \hat{y} ,

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

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

820 This inner product notation gives a clean separation between the *data* (\mathbf{x} and y) and the
 821 *parameters* ($\boldsymbol{\theta}$). This notation also generalizes nicely to the task of **structured prediction**,
 822 which is the focus of Chapters 7-11.

823 It is common to add an **offset feature** at the end of the vector of word counts \mathbf{x} , which
 824 is always 1. We then have to also add an extra zero to each of the zero vectors, to make the
 825 vector lengths match. This gives the entire feature vector $\mathbf{f}(\mathbf{x}, y)$ a length of $(V + 1) \times K$.
 826 The weight associated with this offset feature can be thought of as a bias for or against
 827 each label. For example, if we expect most emails to be spam, then the weight for the
 828 offset feature for $y = \text{SPAM}$ should be larger than the weight for the offset feature for
 829 $y = \text{NOT-SPAM}$.

Returning to the weights $\boldsymbol{\theta}$, where do they come from? One possibility is to set them by hand. If we wanted to distinguish, say, English from Spanish, we can use English and Spanish dictionaries, and set the weight to one for each word that appears in the associated dictionary. For example,³

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

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

²Only $V \times (K - 1)$ features and weights are necessary. By stipulating that $\Psi(\mathbf{x}, y = K) = 0$ regardless of \mathbf{x} , it is possible to implement any classification rule that can be achieved with $V \times K$ features and weights. This is the approach taken in binary classification rules like $y = \text{Sign}(\boldsymbol{\beta} \cdot \mathbf{x} + a)$, where $\boldsymbol{\beta}$ 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 $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$ for all $y \in \mathcal{Y}$.

³In this notation, each tuple (language, word) indexes an element in $\boldsymbol{\theta}$, which remains a vector.

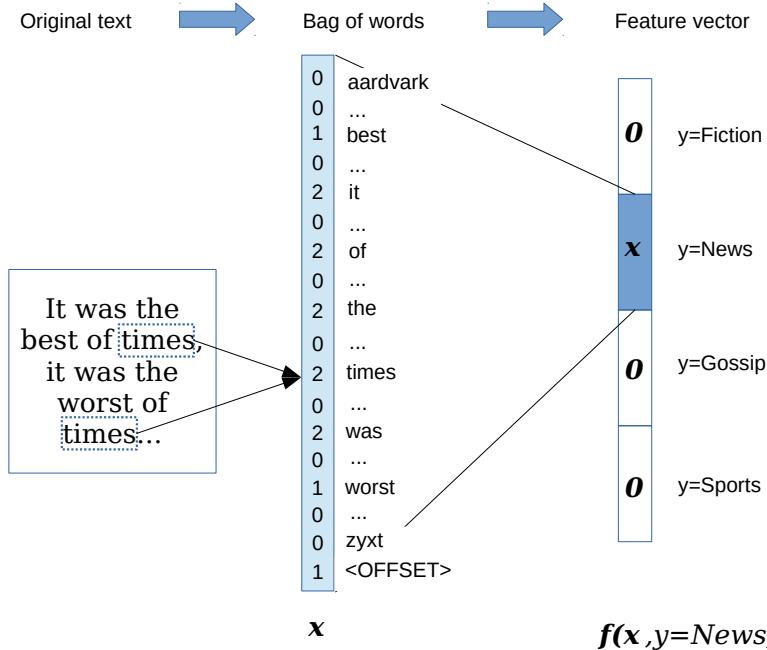


Figure 2.1: The bag-of-words and feature vector representations, for a hypothetical text classification task.

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

840 2.2 Naïve Bayes

841 The **joint probability** of a bag of words x and its true label y is written $p(x, y)$. Suppose
 842 we have a dataset of N labeled instances, $\{(x^{(i)}, y^{(i)})\}_{i=1}^N$, which we assume are **independ-
 843 ent and identically distributed (IID)** (see § A.3). Then the joint probability of the entire
 844 dataset, written $p(x^{(1:N)}, y^{(1:N)})$, is equal to $\prod_{i=1}^N p_{X,Y}(x^{(i)}, y^{(i)})$.⁴

⁴The notation $p_{X,Y}(x^{(i)}, y^{(i)})$ indicates the joint probability that random variables X and Y take the specific values $x^{(i)}$ 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.

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

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

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

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

The notation $p(\mathbf{x}^{(i)}, y^{(i)}; \theta)$ indicates that θ is a *parameter* of the probability function. The product of probabilities can be replaced by a sum of log-probabilities because the log function is monotonically increasing over positive arguments, and so the same θ will maximize both the probability and its logarithm. Working with logarithms is desirable because of numerical stability: on a large dataset, multiplying many probabilities can **underflow** to zero.⁵

The probability $p(\mathbf{x}^{(i)}, y^{(i)}; \theta)$ is defined through a **generative model** — an idealized random process that has generated the observed data.⁶ Algorithm 1 describes the generative model underlying the **Naïve Bayes** classifier, with parameters $\theta = \{\mu, \phi\}$.

- The first line of this generative model encodes the assumption that the instances are mutually independent: neither the label nor the text of document i affects the label or text of document j .⁷ Furthermore, the instances are identically distributed: the distributions over the label $y^{(i)}$ and the text $\mathbf{x}^{(i)}$ (conditioned on $y^{(i)}$) are the same for all instances i . In other words, we make the assumption that every document has the same distribution over labels, and that each document's distribution over words depends only on the label, and not on anything else about the document. We also assume that the documents don't affect each other: if the word *whale* appears in document $i = 7$, that does not make it any more or less likely that it will appear again in document $i = 8$.
- The second line of the generative model states that the random variable $y^{(i)}$ is drawn from a categorical distribution with parameter μ . Categorical distributions are like

⁵Throughout 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.

⁶Generative 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).

⁷Can you think of any cases in which this assumption is too strong?

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

866 weighted dice: the column vector $\boldsymbol{\mu} = [\mu_1; \mu_2; \dots; \mu_K]$ gives the probabilities of
 867 each label, so that the probability of drawing label y is equal to μ_y . For example, if
 868 $\mathcal{Y} = \{\text{POSITIVE}, \text{NEGATIVE}, \text{NEUTRAL}\}$, we might have $\boldsymbol{\mu} = [0.1; 0.7; 0.2]$. We require
 869 $\sum_{y \in \mathcal{Y}} \mu_y = 1$ and $\mu_y \geq 0, \forall y \in \mathcal{Y}$: each label's probability is non-negative, and the
 870 sum of these probabilities is equal to one.⁸

- 871 • The third line describes how the bag-of-words counts $\mathbf{x}^{(i)}$ are generated. By writing
 872 $\mathbf{x}^{(i)} | y^{(i)}$, this line indicates that the word counts are conditioned on the label, so
 873 that the joint probability is factored using the chain rule,

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

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

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

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

874 The term $B(\mathbf{x})$ is called the **multinomial coefficient**.

875 As in the categorical distribution, the parameter ϕ_j can be interpreted as a probability:
 876 specifically, the probability that any given token in the document is the word j .
 877 The multinomial distribution involves a product over words, with each term in the
 878 product equal to the probability ϕ_j , exponentiated by the count x_j . Words that have
 879 zero count play no role in this product, because $\phi_j^0 = 1$. The multinomial coefficient
 880 $B(\mathbf{x})$ doesn't depend on $\boldsymbol{\phi}$, and can usually be ignored. Can you see why we need
 881 this term at all?⁹

⁸Formally, we require $\boldsymbol{\mu} \in \Delta^{K-1}$, where Δ^{K-1} is the $K - 1$ **probability simplex**, the set of all vectors of K nonnegative numbers that sum to one. Because of the sum-to-one constraint, there are $K - 1$ degrees of freedom for a vector of size K .

⁹Technically, a multinomial distribution requires a second parameter, the total number of word counts in \mathbf{x} . In the bag-of-words representation is equal to the number of words in the document. However, this parameter is irrelevant for classification.

Algorithm 2 Alternative generative process for the Naïve Bayes classifier

```

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

```

882 The notation $p(x | y; \phi)$ indicates the conditional probability of word counts x
 883 given label y , with parameter ϕ , which is equal to $p_{\text{mult}}(x; \phi_y)$. By specifying the
 884 multinomial distribution, we describe the **multinomial Naïve Bayes** classifier. Why
 885 “naïve”? Because the multinomial distribution treats each word token indepen-
 886 dently, conditioned on the class: the probability mass function factorizes across the
 887 counts.¹⁰

888 **2.2.1 Types and tokens**

889 A slight modification to the generative model of Naïve Bayes is shown in Algorithm 2.
 890 Instead of generating a vector of counts of **types**, x , this model generates a *sequence* of
 891 **tokens**, $w = (w_1, w_2, \dots, w_M)$. The distinction between types and tokens is critical: $x_j \in$
 892 $\{0, 1, 2, \dots, M\}$ is the count of word type j in the vocabulary, e.g., the number of times
 893 the word *cannibal* appears; $w_m \in \mathcal{V}$ is the identity of token m in the document, e.g. $w_m =$
 894 *cannibal*.

895 The probability of the sequence w is a product of categorical probabilities. Algorithm 2
 896 makes a conditional independence assumption: each token $w_m^{(i)}$ is independent of all other
 897 tokens $w_{n \neq m}^{(i)}$, conditioned on the label $y^{(i)}$. This is identical to the “naïve” independence
 898 assumption implied by the multinomial distribution, and as a result, the optimal parame-
 899 ters for this model are identical to those in multinomial Naïve Bayes. For any instance, the
 900 probability assigned by this model is proportional to the probability under multinomial
 901 Naïve Bayes. The constant of proportionality is the multinomial coefficient $B(x)$. Because
 902 $B(x) \geq 1$, the probability for a vector of counts x is at least as large as the probability
 903 for a list of words w that induces the same counts: there can be many word sequences
 904 that correspond to a single vector of counts. For example, *man bites dog* and *dog bites man*
 905 correspond to an identical count vector, $\{bites : 1, dog : 1, man : 1\}$, and $B(x)$ is equal to
 906 the total number of possible word orderings for count vector x .

907 Sometimes it is useful to think of instances as counts of types, x ; other times, it is
 908 better to think of them as sequences of tokens, w . If the tokens are generated from a

¹⁰You 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.

model that assumes conditional independence, then these two views lead to probability models that are identical, except for a scaling factor that does not depend on the label or the parameters.

2.2.2 Prediction

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

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

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

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

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

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

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

where

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

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

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

2.2.3 Estimation

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

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

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

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

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

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

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

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

Maximum-likelihood estimation chooses ϕ to maximize the log-likelihood \mathcal{L} . However, the solution must obey the following constraints:

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

These constraints can be incorporated by adding a set of Lagrange multipliers to the objective (see Appendix B for more details). To solve for each θ_y , we maximize the Lagrangian,

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

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

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

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

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

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

933 where $\delta(y^{(i)} = y)$ is a **delta function**, also sometimes called an indicator function, which
 934 returns one if $y^{(i)} = y$. The symbol \propto indicates that $\phi_{y,j}$ is **proportional** to the right-hand
 935 side of the equation.

Equation 2.28 shows three different notations for the same thing: a sum over the word counts for all documents i such that the label $y^{(i)} = y$. This gives a solution for each ϕ_y up to a constant of proportionality. Now recall the constraint $\sum_{j=1}^V \phi_{y,j} = 1$, which arises because ϕ_y represents a vector of probabilities for each word in the vocabulary. This constraint leads to an exact solution, which does not depend on λ :

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

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

938 2.2.4 Smoothing

939 With text data, there are likely to be pairs of labels and words that never appear in the
 940 training set, leaving $\phi_{y,j} = 0$. For example, the word *molybdenum* may have never yet
 941 appeared in a work of fiction. But choosing a value of $\phi_{\text{FICTION}, \text{molybdenum}} = 0$ would allow
 942 this single feature to completely veto a label, since $p(\text{FICTION} | \mathbf{x}) = 0$ if $x_{\text{molybdenum}} > 0$.

943 This is undesirable, because it imposes high **variance**: depending on what data happens to be in the training set, we could get vastly different classification rules. One solution is to **smooth** the probabilities, by adding a “pseudocount” of α to each count, and then normalizing.

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

947 This is called **Laplace smoothing**.¹¹ The pseudocount α is a **hyperparameter**, because it
 948 controls the form of the log-likelihood function, which in turn drives the estimation of ϕ .

949 Smoothing reduces variance, but moves us away from the maximum likelihood estimate: it imposes a **bias**. In this case, the bias points towards uniform probabilities. Machine learning theory shows that errors on heldout data can be attributed to the sum of bias and variance (Mohri et al., 2012). In general, techniques for reducing variance often increase the bias, leading to a **bias-variance tradeoff**.

¹¹Laplace smoothing has a Bayesian justification, in which the generative model is extended to include ϕ as a random variable. The resulting distribution over ϕ depends on both the data (\mathbf{x} and y) and the **prior probability** $p(\phi; \alpha)$. The corresponding estimate of ϕ is called **maximum a posteriori**, or MAP. This is in contrast with maximum likelihood, which depends only on the data.

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

959 Similar issues arise throughout machine learning. Later in this chapter we will encounter
960 **regularization**, which controls the bias-variance tradeoff for logistic regression and large-
961 margin classifiers (§ 2.5.1); § 3.3.2 describes techniques for controlling variance in deep
962 learning; chapter 6 describes more elaborate methods for smoothing empirical probabili-
963 ties.

964 2.2.5 Setting hyperparameters

965 Returning to Naïve Bayes, how should we choose the best value of hyperparameters like
966 α ? Maximum likelihood will not work: the maximum likelihood estimate of α on the
967 training set will always be $\alpha = 0$. In many cases, what we really want is **accuracy**: the
968 number of correct predictions, divided by the total number of predictions. (Other mea-
969 sures of classification performance are discussed in § 4.4.) As we will see, it is hard to
970 optimize for accuracy directly. But for scalar hyperparameters like α can be tuned by a
971 simple heuristic called **grid search**: try a set of values (e.g., $\alpha \in \{0.001, 0.01, 0.1, 1, 10\}$),
972 compute the accuracy for each value, and choose the setting that maximizes the accuracy.

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

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

989 When only a small amount of labeled data is available, the test set accuracy can be
990 unreliable. *K*-fold **cross-validation** is one way to cope with this scenario: the labeled

991 data is divided into K folds, and each fold acts as the test set, while training on the other
 992 folds. The test set accuracies are then aggregated. In the extreme, each fold is a single data
 993 point; this is called **leave-one-out cross-validation**. To perform hyperparameter tuning
 994 in the context of cross-validation, another fold can be used for grid search. It is important
 995 not to repeatedly evaluate the cross-validated accuracy while making design decisions
 996 about the classifier, or you will overstate the accuracy on truly unseen data.

997 2.3 Discriminative learning

998 Naïve Bayes is easy to work with: the weights can be estimated in closed form, and the
 999 probabilistic interpretation makes it relatively easy to extend. However, the assumption
 1000 that features are independent can seriously limit its accuracy. Thus far, we have defined
 1001 the feature function $f(x, y)$ so that it corresponds to bag-of-words features: one feature
 1002 per word in the vocabulary. In natural language, bag-of-words features violate the as-
 1003 sumption of conditional independence — for example, the probability that a document
 1004 will contain the word *naïve* is surely higher given that it also contains the word *Bayes* —
 1005 but this violation is relatively mild.

1006 However, good performance on text classification often requires features that are richer
 1007 than the bag-of-words:

- 1008 • To better handle out-of-vocabulary terms, we want features that apply to multiple
 1009 words, such as prefixes and suffixes (e.g., *anti-*, *un-*, *-ing*) and capitalization.
- 1010 • We also want n -gram features that apply to multi-word units: **bigrams** (e.g., *not*
 1011 *good*, *not bad*), **trigrams** (e.g., *not so bad*, *lacking any decency*, *never before imagined*), and
 1012 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:¹²

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

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

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

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

¹²The 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).

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

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

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

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

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

2.3.1 Perceptron

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

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

The perceptron algorithm may seem like an unprincipled heuristic: Naïve Bayes has a solid foundation in probability, but the perceptron is just adding and subtracting constants from the weights every time there is a mistake. Will this really work? In fact, there is some nice theory for the perceptron, based on the concept of **linear separability**. Informally, a dataset with binary labels ($y \in \{0, 1\}$) is linearly separable if it is possible to draw a hyperplane (a line in many dimensions), such that on each side of the hyperplane, all instances have the same label. This definition can be formalized and extended to multiple labels:

Definition 1 (Linear separability). *The dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$ is linearly separable iff (if and only if) there exists some weight vector $\boldsymbol{\theta}$ and some margin ρ such that for every instance*

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

```

1045 $(\mathbf{x}^{(i)}, y^{(i)})$, 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
 1046 at least ρ greater than inner product of $\boldsymbol{\theta}$ and the feature function for every other possible label,
 1047 $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y')$.

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

1048 Linear separability is important because of the following guarantee: if your data is
 1049 linearly separable, then the perceptron algorithm will find a separator (Novikoff, 1962).¹³
 1050 So while the perceptron may seem heuristic, it is guaranteed to succeed, if the learning
 1051 problem is easy enough.

1052 How useful is this proof? Minsky and Papert (1969) famously proved that the simple
 1053 logical function of *exclusive-or* is not separable, and that a perceptron is therefore inca-
 1054 pable of learning this function. But this is not just an issue for the perceptron: any linear
 1055 classification algorithm, including Naïve Bayes, will fail on this task. Text classifica-
 1056 tion problems usually involve high dimensional feature spaces, with thousands or millions of
 1057 features. For these problems, it is very likely that the training data is indeed separable.
 1058 And even if the dataset is not separable, it is still possible to place an upper bound on the
 1059 number of errors that the perceptron algorithm will make (Freund and Schapire, 1999).

¹³It is also possible to prove an upper bound on the number of training iterations required to find the separator. Proofs like this are part of the field of **machine learning theory** (Mohri et al., 2012).

Algorithm 4 Averaged perceptron learning algorithm

```

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

```

1060 **2.3.2 Averaged perceptron**

1061 The perceptron iterates over the data repeatedly — until “tired”, as described in Algo-
 1062 rithm 3. If the data is linearly separable, the perceptron will eventually find a separator,
 1063 and we can stop once all training instances are classified correctly. But if the data is not
 1064 linearly separable, the perceptron can *thrash* between two or more weight settings, never
 1065 converging. In this case, how do we know that we can stop training, and how should
 1066 we choose the final weights? An effective practical solution is to *average* the perceptron
 1067 weights across all iterations.

1068 This procedure is shown in Algorithm 4. The learning algorithm is nearly identical,
 1069 but we also maintain a vector of the sum of the weights, \mathbf{m} . At the end of the learning
 1070 procedure, we divide this sum by the total number of updates t , to compute the average
 1071 weights, $\bar{\boldsymbol{\theta}}$. These average weights are then used for prediction. In the algorithm sketch,
 1072 the average is computed from a running sum, $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}$. However, this is inefficient,
 1073 because it requires $|\boldsymbol{\theta}|$ operations to update the running sum. When $\mathbf{f}(\mathbf{x}, y)$ is sparse,
 1074 $|\boldsymbol{\theta}| \gg |\mathbf{f}(\mathbf{x}, y)|$ for any individual (\mathbf{x}, y) . This means that computing the running sum will
 1075 be much more expensive than computing the update to $\boldsymbol{\theta}$ itself, which requires only
 1076 $2 \times |\mathbf{f}(\mathbf{x}, y)|$ operations. One of the exercises is to sketch a more efficient algorithm for
 1077 computing the averaged weights.

1078 Even if the dataset is not separable, the averaged weights will eventually converge.
 1079 One possible stopping criterion is to check the difference between the average weight
 1080 vectors after each pass through the data: if the norm of the difference falls below some

1081 predefined threshold, we can stop training. Another stopping criterion is to hold out some
 1082 data, and to measure the predictive accuracy on this heldout data. When the accuracy
 1083 on the heldout data starts to decrease, the learning algorithm has begun to **overfit** the
 1084 training set. At this point, it is probably best to stop; this stopping criterion is known as
 1085 **early stopping**.

1086 **Generalization** is the ability to make good predictions on instances that are not in
 1087 the training data. Averaging can be proven to improve generalization, by computing an
 1088 upper bound on the generalization error (Freund and Schapire, 1999; Collins, 2002).

1089 2.4 Loss functions and large-margin classification

1090 Naïve Bayes chooses the weights θ by maximizing the joint log-likelihood $\log p(\mathbf{x}^{(1:N)}, y^{(1:N)})$.
 1091 By convention, optimization problems are generally formulated as minimization of a **loss**
 1092 **function**. The input to a loss function is the vector of weights θ , and the output is a
 1093 non-negative number, measuring the performance of the classifier on a training instance.
 1094 Formally, the loss $\ell(\theta; \mathbf{x}^{(i)}, y^{(i)})$ is then a measure of the performance of the weights θ on
 1095 the instance $(\mathbf{x}^{(i)}, y^{(i)})$. The goal of learning is to minimize the sum of the losses across all
 1096 instances in the training set.

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

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

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

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

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

1097 The problem of minimizing ℓ_{NB} is thus identical to maximum-likelihood estimation.

1098 Loss functions provide a general framework for comparing learning objectives. For
 1099 example, an alternative loss function is the **zero-one loss**,

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

1100 The zero-one loss is zero if the instance is correctly classified, and one otherwise. The
 1101 sum of zero-one losses is proportional to the error rate of the classifier on the training

1102 data. Since a low error rate is often the ultimate goal of classification, this may seem
 1103 ideal. But the zero-one loss has several problems. One is that it is **non-convex**,¹⁴ which
 1104 means that there is no guarantee that gradient-based optimization will be effective. A
 1105 more serious problem is that the derivatives are useless: the partial derivative with respect
 1106 to any parameter is zero everywhere, except at the points where $\theta \cdot f(\mathbf{x}^{(i)}, y) = \theta \cdot f(\mathbf{x}^{(i)}, \hat{y})$
 1107 for some \hat{y} . At those points, the loss is discontinuous, and the derivative is undefined.

1108 The perceptron optimizes a loss function that has better properties for learning:

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

1109 When $\hat{y} = y^{(i)}$, the loss is zero; otherwise, it increases linearly with the gap between the score for the predicted label \hat{y} and the score for the true label $y^{(i)}$. Plotting this loss against
 1110 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
 1111 **hinge loss**.

1113 To see why this is the loss function optimized by the perceptron, take the derivative
 1114 with respect to $\boldsymbol{\theta}$,

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

1115 At each instance, the perceptron algorithm takes a step of magnitude one in the opposite
 1116 direction of this **gradient**, $\nabla_{\boldsymbol{\theta}} \ell_{\text{PERCEPTRON}} = \frac{\partial}{\partial \boldsymbol{\theta}} \ell_{\text{PERCEPTRON}}(\boldsymbol{\theta}; \mathbf{x}^{(i)}, y^{(i)})$. As we will see in
 1117 § 2.6, this is an example of the optimization algorithm **stochastic gradient descent**, applied
 1118 to the objective in Equation 2.41.

1119 ***Breaking ties with subgradient descent** Careful readers will notice the tacit assumption
 1120 that there is a unique \hat{y} that maximizes $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$. What if there are two or more
 1121 labels that maximize this function? Consider binary classification: if the maximizer is $y^{(i)}$,
 1122 then the gradient is zero, and so is the perceptron update; if the maximizer is $\hat{y} \neq y^{(i)}$,
 1123 then the update is the difference $\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) - \mathbf{f}(\mathbf{x}^{(i)}, \hat{y})$. The underlying issue is that the
 1124 perceptron loss is not **smooth**, because the first derivative has a discontinuity at the hinge
 1125 point, where the score for the true label $y^{(i)}$ is equal to the score for some other label \hat{y} . At
 1126 this point, there is no unique gradient; rather, there is a set of **subgradients**. A vector \mathbf{v} is
 1127 a subgradient of the function g at \mathbf{u}_0 iff $g(\mathbf{u}) - g(\mathbf{u}_0) \geq \mathbf{v} \cdot (\mathbf{u} - \mathbf{u}_0)$ for all \mathbf{u} . Graphically,
 1128 this defines the set of hyperplanes that include $g(\mathbf{u}_0)$ and do not intersect g at any other
 1129 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

¹⁴A 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.

approach from the right, the gradient is 0. At the hinge point, the subgradients include all vectors that are bounded by these two extremes. In subgradient descent, *any* subgradient can be used (Bertsekas, 2012). Since both $\mathbf{0}$ and $\mathbf{f}(\mathbf{x}, \hat{y}) - \mathbf{f}(\mathbf{x}, y)$ are subgradients at the hinge point, either one can be used in the perceptron update.

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

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

2.4.1 Large margin classification

This last comment suggests a potential problem with the perceptron. Suppose a test example is very close to a training example, but not identical. If the classifier only gets the correct answer on the training example by a small margin, then it may give a different answer on the nearby test instance. 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 incorrect label. The intuition behind **large margin classification** is that it is not enough 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,

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

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

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

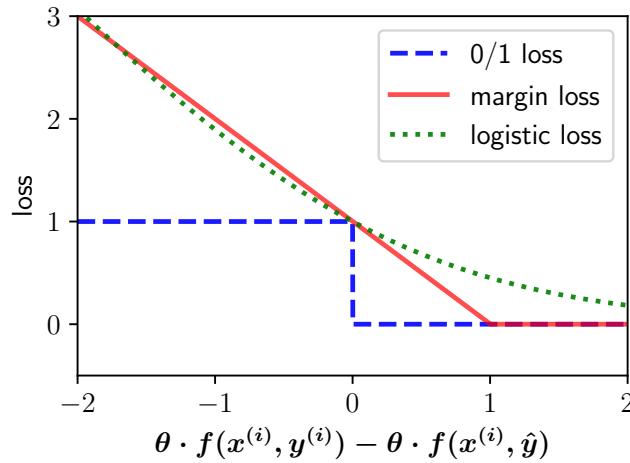


Figure 2.2: Margin, zero-one, and logistic loss functions.

1153 **2.4.2 Support vector machines**

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

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

1154 This is a **constrained optimization** problem, where the second line describes constraints
 1155 on the space of possible solutions θ . In this case, the constraint is that the functional
 1156 margin always be at least one, and the objective is that the minimum geometric margin
 1157 be as large as possible.

A property of the norm $\|\theta\|_2$ is that it scales linearly: $\|a\theta\|_2 = a\|\theta\|_2$. Furthermore,



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.

the functional margin γ is a linear function of θ , so that $\gamma(a\theta, \mathbf{x}^{(i)}, y^{(i)}) = a\gamma(\theta, \mathbf{x}^{(i)}, y^{(i)})$. As a result, any scaling factor on θ will cancel in the numerator and denominator of the geometric margin. If the data is linearly separable at any $\rho > 0$, it is always possible to rescale the functional margin to 1 by multiplying θ by a scalar constant. 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 $\frac{1}{2}\|\theta\|_2^2 = \frac{1}{2}\sum\theta_j^2$, which is easier to work with. This gives the optimization problem,

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

1158 This optimization problem is a **quadratic program**: the objective is a quadratic function
 1159 of the parameters, and the constraints are all linear inequalities. The resulting clas-
 1160 sifier is better known as the **support vector machine**. The name derives from one of the
 1161 solutions, which is to incorporate the constraints through Lagrange multipliers $\alpha_i \geq 0, i =$
 1162 $1, 2, \dots, N$. The instances for which $\alpha_i > 0$ are the **support vectors**; other instances are
 1163 irrelevant to the classification boundary.

1164 **2.4.3 Slack variables**

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

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

1165 The hyperparameter C controls the tradeoff between violations of the margin constraint and the preference for a low norm of θ . As $C \rightarrow \infty$, slack is infinitely expensive, 1166 and there is only a solution if the data is separable. As $C \rightarrow 0$, slack becomes free, and 1167 there is a trivial solution at $\theta = 0$. Thus, C plays a similar role to the smoothing parameter 1168 in Naïve Bayes (§ 2.2.4), trading off between a close fit to the training data and better 1169 generalization. Like the smoothing parameter of Naïve Bayes, C must be set by the user, 1170 typically by maximizing performance on a heldout development set.

1172 To solve the constrained optimization problem defined in Equation 2.48, we can first 1173 solve for the slack variables,

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

The inequality is tight: the optimal solution is to make the slack variables as small as possible, while still satisfying the constraints (Ratliff et al., 2007; Smith, 2011). By plugging in the minimum slack variables back into Equation 2.48, the problem can be transformed into the unconstrained optimization,

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

1174 where each ξ_i has been substituted by the right-hand side of Equation 2.49, and the factor 1175 of C on the slack variables has been replaced by an equivalent factor of $\lambda = \frac{1}{C}$ on the 1176 norm of the weights.

1177 Now define the **cost** of a classification error as,¹⁵

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

¹⁵We can also define specialized cost functions that heavily penalize especially undesirable errors (Tsodkatidis et al., 2004). This idea is revisited in chapter 7.

Equation 2.50 can be rewritten using this cost function,

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

1178 This objective maximizes over all $y \in \mathcal{Y}$, in search of labels that are both *strong*, as measured by $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y)$, and *wrong*, as measured by $c(y^{(i)}, y)$. This maximization is known
 1179 as **cost-augmented decoding**, because it augments the maximization objective to favor
 1180 high-cost predictions. If the highest-scoring label is $y = y^{(i)}$, then the margin constraint
 1181 is satisfied, and the loss for this instance is zero. Cost augmentation is only done while
 1182 learning; it is not applied when making predictions on unseen data.

Differentiating Equation 2.52 with respect to the weights gives,

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

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

1184 where L_{SVM} refers to minimization objective in Equation 2.52. This gradient is very similar
 1185 to the perceptron update. One difference is the additional term $\lambda \boldsymbol{\theta}$, which **regularizes** the
 1186 weights towards 0. The other difference is the cost $c(y^{(i)}, y)$, which is added to $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$
 1187 when choosing \hat{y} during training. This term derives from the margin constraint: large
 1188 margin classifiers learn not only from instances that are incorrectly classified, but also
 1189 from instances for which the correct classification decision was not sufficiently confident.

1190 2.5 Logistic regression

1191 Thus far, we have seen two broad classes of learning algorithms. Naïve Bayes is a prob-
 1192 abilistic method, where learning is equivalent to estimating a joint probability distribu-
 1193 tion. The perceptron and support vector machine are discriminative, error-driven algo-
 1194 rithms: the learning objective is closely related to the number of errors on the training
 1195 data. Probabilistic and error-driven approaches each have advantages: probability makes
 1196 it possible to quantify uncertainty about the predicted labels, but the probability model of
 1197 Naïve Bayes makes unrealistic independence assumptions that limit the features that can
 1198 be used.

Logistic regression combines advantages of discriminative and probabilistic classi-
 1199 fiers. Unlike Naïve Bayes, which starts from the **joint probability** $p_{X,Y}$, logistic regression
 defines the desired **conditional probability** $p_{Y|X}$ directly. Think of $\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$ as a scoring
 1200 function for the compatibility of the base features \mathbf{x} and the label y . To convert this score
 1201 into a probability, we first exponentiate, obtaining $\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y))$, which is guaranteed

to be non-negative. Next, we normalize, dividing over all possible labels $y' \in \mathcal{Y}$. The resulting conditional probability is defined as,

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

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

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

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

1199 The final line is obtained by plugging in Equation 2.55 and taking the logarithm.¹⁶ Inside
1200 the sum, we have the (additive inverse of the) **logistic loss**,

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

1201 The logistic loss is shown in Figure 2.2. A key difference from the zero-one and hinge
1202 losses is that logistic loss is never zero. This means that the objective function can always
1203 be improved by assigning higher confidence to the correct label.

1204 2.5.1 Regularization

1205 As with the support vector machine, better generalization can be obtained by penalizing
1206 the norm of $\boldsymbol{\theta}$. This is done by adding a term of $\frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$ to the minimization objective.
1207 This is called L_2 regularization, because $\|\boldsymbol{\theta}\|_2^2$ is the squared L_2 norm of the vector $\boldsymbol{\theta}$.
1208 Regularization forces the estimator to trade off performance on the training data against
1209 the norm of the weights, and this can help to prevent overfitting. Consider what would
1210 happen to the unregularized weight for a base feature j that is active in only one instance
1211 $\mathbf{x}^{(i)}$: the conditional log-likelihood could always be improved by increasing the weight
1212 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
1213 feature associated with $x_j^{(i)}$ and label y in $\mathbf{f}(\mathbf{x}^{(i)}, y)$.

In § 2.2.4, 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

¹⁶The 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.

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 θ . The log-likelihood under a zero-mean Gaussian is,

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

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

2.5.2 Gradients

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

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

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

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

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

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

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

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

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

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

1224 **2.6 Optimization**

1225 Each of the classification algorithms in this chapter can be viewed as an optimization
 1226 problem:

- 1227 • In Naïve Bayes, the objective is the joint likelihood $\log p(\mathbf{x}^{(1:N)}, \mathbf{y}^{(1:N)})$. Maximum
 1228 likelihood estimation yields a closed-form solution for $\boldsymbol{\theta}$.
- 1229 • In the support vector machine, the objective is the regularized margin loss,

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

1230 There is no closed-form solution, but the objective is convex. The perceptron algo-
 1231 rithm minimizes a similar objective.

- 1232 • In logistic regression, the objective is the regularized negative log-likelihood,

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

1233 Again, there is no closed-form solution, but the objective is convex.

1234 These learning algorithms are distinguished by *what* is being optimized, rather than
 1235 *how* the optimal weights are found. This decomposition is an essential feature of con-
 1236 temporary machine learning. The domain expert's job is to design an objective function
 1237 — or more generally, a **model** of the problem. If the model has certain characteristics,
 1238 then generic optimization algorithms can be used to find the solution. In particular, if an
 1239 objective function is differentiable, then gradient-based optimization can be employed;
 1240 if it is also convex, then gradient-based optimization is guaranteed to find the globally
 1241 optimal solution. The support vector machine and logistic regression have both of these
 1242 properties, and so are amenable to generic **convex optimization** techniques (Boyd and
 1243 Vandenberghe, 2004).

1244 **2.6.1 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,

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

1245 where $\nabla_{\theta} L$ is the gradient computed over the entire training set, and $\eta^{(t)}$ is the **learning**
 1246 **rate** at iteration t . If the objective L is a convex function of θ , then this procedure is
 1247 guaranteed to terminate at the global optimum, for appropriate schedule of learning rates,
 1248 $\eta^{(t)}$.¹⁷

1249 In practice, gradient descent can be slow to converge, as the gradient can become
 1250 infinitesimally small. Faster convergence can be obtained by second-order Newton opti-
 1251 mization, which incorporates the inverse of the **Hessian matrix**,

$$H_{i,j} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j} \quad [2.70]$$

1252 The size of the Hessian matrix is quadratic in the number of features. In the bag-of-words
 1253 representation, this is usually too big to store, let alone invert. **Quasi-Network optimiza-**
 1254 **tion** techniques maintain a low-rank approximation to the inverse of the Hessian matrix.
 1255 Such techniques usually converge more quickly than gradient descent, while remaining
 1256 computationally tractable even for large feature sets. A popular quasi-Newton algorithm
 1257 is L-BFGS (Liu and Nocedal, 1989), which is implemented in many scientific computing
 1258 environments, such as SCIPY and MATLAB.

1259 For any gradient-based technique, the user must set the learning rates $\eta^{(t)}$. While con-
 1260 vergence proofs usually employ a decreasing learning rate, in practice, it is common to fix
 1261 $\eta^{(t)}$ to a small constant, like 10^{-3} . The specific constant can be chosen by experimentation,
 1262 although there is research on determining the learning rate automatically (Schaul et al.,
 1263 2013; Wu et al., 2018).

1264 2.6.2 Online optimization

1265 Batch optimization computes the objective on the entire training set before making an up-
 1266 date. This may be inefficient, because at early stages of training, a small number of train-
 1267 ing examples could point the learner in the correct direction. **Online learning** algorithms
 1268 make updates to the weights while iterating through the training data. The theoretical
 1269 basis for this approach is a stochastic approximation to the true objective function,

$$\sum_{i=1}^N \ell(\theta; \mathbf{x}^{(i)}, y^{(i)}) \approx N \times \ell(\theta; \mathbf{x}^{(j)}, y^{(j)}), \quad (\mathbf{x}^{(j)}, y^{(j)}) \sim \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N, \quad [2.71]$$

1270 where the instance $(\mathbf{x}^{(j)}, y^{(j)})$ is sampled at random from the full dataset.

1271 In **stochastic gradient descent**, the approximate gradient is computed by randomly
 1272 sampling a single instance, and an update is made immediately. This is similar to the

¹⁷Convergence proofs typically require the learning rate to satisfy the following conditions: $\sum_{t=1}^{\infty} \eta^{(t)} = \infty$ and $\sum_{t=1}^{\infty} (\eta^{(t)})^2 < \infty$ (Bottou et al., 2016). These properties are satisfied by any 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  $\boldsymbol{\theta}^{(t)}$ 

```

1273 perceptron algorithm, which also updates the weights one instance at a time. In **mini-**
 1274 **batch** stochastic gradient descent, the gradient is computed over a small set of instances.
 1275 A typical approach is to set the minibatch size so that the entire batch fits in memory on a
 1276 graphics processing unit (GPU; Neubig et al., 2017). It is then possible to speed up learn-
 1277 ing by parallelizing the computation of the gradient over each instance in the minibatch.

1278 Algorithm 5 offers a generalized view of gradient descent. In standard gradient de-
 1279 scent, the batcher returns a single batch with all the instances. In stochastic gradient de-
 1280 scent, it returns N batches with one instance each. In mini-batch settings, the batcher
 1281 returns B minibatches, $1 < B < N$.

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

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}; \mathbf{x}^{(i)}, y^{(i)}) \quad [2.72]$$

$$\boldsymbol{\theta}_j^{(t+1)} \leftarrow \boldsymbol{\theta}_j^{(t)} - \frac{\eta^{(t)}}{\sqrt{\sum_{t'=1}^t g_{t,j}^2}} g_{t,j}, \quad [2.73]$$

1282 where j iterates over features in $f(\mathbf{x}, y)$.

1283 In most cases, the number of active features for any instance is much smaller than the
 1284 number of weights. If so, the computation cost of online optimization will be dominated
 1285 by the update from the regularization term, $\lambda\theta$. The solution is to be “lazy”, updating
 1286 each θ_j only as it is used. To implement lazy updating, store an additional parameter τ_j ,
 1287 which is the iteration at which θ_j was last updated. If θ_j is needed at time t , the $t - \tau$
 1288 regularization updates can be performed all at once. This strategy is described in detail
 1289 by Kummerfeld et al. (2015).

1290 2.7 *Additional topics in classification

1291 Throughout this text, advanced topics will be marked with an asterisk.

1292 2.7.1 Feature selection by regularization

1293 In logistic regression and large-margin classification, generalization can be improved by
 1294 regularizing the weights towards 0, using the L_2 norm. But rather than encouraging
 1295 weights to be small, it might be better for the model to be **sparse**: it should assign weights
 1296 of exactly zero to most features, and only assign non-zero weights to features that are
 1297 clearly necessary. This idea can be formalized by the L_0 norm, $L_0 = \|\theta\|_0 = \sum_j \delta(\theta_j \neq 0)$,
 1298 which applies a constant penalty for each non-zero weight. This norm can be thought
 1299 of as a form of **feature selection**: optimizing the L_0 -regularized conditional likelihood is
 1300 equivalent to trading off the log-likelihood against the number of active features. Reduc-
 1301 ing the number of active features is desirable because the resulting model will be fast,
 1302 low-memory, and should generalize well, since irrelevant features will be pruned away.
 1303 Unfortunately, the L_0 norm is non-convex and non-differentiable. Optimization under L_0
 1304 regularization is **NP-hard**, meaning that it can be solved efficiently only if P=NP (Ge et al.,
 1305 2011).

1306 A useful alternative is the L_1 norm, which is equal to the sum of the absolute values
 1307 of the weights, $\|\theta\|_1 = \sum_j |\theta_j|$. The L_1 norm is convex, and can be used as an approxima-
 1308 tion to L_0 (Tibshirani, 1996). Conveniently, the L_1 norm also performs feature selection,
 1309 by driving many of the coefficients to zero; it is therefore known as a **sparsity inducing**
 1310 **regularizer**. The L_1 norm does not have a gradient at $\theta_j = 0$, so we must instead optimize
 1311 the L_1 -regularized objective using **subgradient** methods. The associated stochastic sub-
 1312 gradient descent algorithms are only somewhat more complex than conventional SGD;
 1313 Sra et al. (2012) survey approaches for estimation under L_1 and other regularizers.

1314 Gao et al. (2007) compare L_1 and L_2 regularization on a suite of NLP problems, finding
 1315 that L_1 regularization generally gives similar accuracy to L_2 regularization, but that L_1
 1316 regularization produces models that are between ten and fifty times smaller, because more
 1317 than 90% of the feature weights are set to zero.

1318 **2.7.2 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**,

$$\sigma(a) \triangleq \frac{\exp(a)}{1 + \exp(a)} = (1 + \exp(-a))^{-1} \quad [2.74]$$

$$p(y | x; \theta) = \sigma(\theta \cdot x). \quad [2.75]$$

1319 This is the origin of the name “logistic regression.” Logistic regression can be viewed as
 1320 part of a larger family of **generalized linear models** (GLMs), in which various other **link**
 1321 **functions** convert between the inner product $\theta \cdot x$ and the parameter of a conditional
 1322 probability distribution.

1323 Logistic regression and related models are sometimes referred to as **log-linear**, be-
 1324 cause the log-probability is a linear function of the features. But in the early NLP liter-
 1325 ature, logistic regression was often called **maximum entropy** classification (Berger et al.,
 1326 1996). This name refers to an alternative formulation, in which the goal is to find the max-
 1327 imum entropy probability function that satisfies **moment-matching** constraints. These
 1328 constraints specify that the empirical counts of each feature should match the expected
 1329 counts under the induced probability distribution $p_{Y|X;\theta}$,

$$\sum_{i=1}^N f_j(x^{(i)}, y^{(i)}) = \sum_{i=1}^N \sum_{y \in \mathcal{Y}} p(y | x^{(i)}; \theta) f_j(x^{(i)}, y), \quad \forall j \quad [2.76]$$

1330 The moment-matching constraint is satisfied exactly when the derivative of the condi-
 1331 tional log-likelihood function (Equation 2.64) is equal to zero. However, the constraint
 1332 can be met by many values of θ , so which should we choose?

1333 The **entropy** of the conditional probability distribution $p_{Y|X}$ is,

$$H(p_{Y|X}) = - \sum_{x \in \mathcal{X}} p_X(x) \sum_{y \in \mathcal{Y}} p_{Y|X}(y | x) \log p_{Y|X}(y | x), \quad [2.77]$$

1334 where \mathcal{X} is the set of all possible feature vectors, and $p_X(x)$ is the probability of observing
 1335 the base features x . The distribution p_X is unknown, but it can be estimated by summing
 1336 over all the instances in the training set,

$$\tilde{H}(p_{Y|X}) = - \frac{1}{N} \sum_{i=1}^N \sum_{y \in \mathcal{Y}} p_{Y|X}(y | x^{(i)}) \log p_{Y|X}(y | x^{(i)}). \quad [2.78]$$

1337 If the entropy is large, the likelihood function is smooth across possible values of y ;
 1338 if it is small, the likelihood function is sharply peaked at some preferred value; in the

1339 limiting case, the entropy is zero if $p(y | x) = 1$ for some y . The maximum-entropy cri-
 1340 terion chooses to make the weakest commitments possible, while satisfying the moment-
 1341 matching constraints from Equation 2.76. The solution to this constrained optimization
 1342 problem is identical to the maximum conditional likelihood (logistic-loss) formulation
 1343 that was presented in § 2.5.

1344 2.8 Summary of learning algorithms

1345 It is natural to ask which learning algorithm is best, but the answer depends on what
 1346 characteristics are important to the problem you are trying to solve.

1347 **Naïve Bayes** *Pros:* easy to implement; estimation is fast, requiring only a single pass over
 1348 the data; assigns probabilities to predicted labels; controls overfitting with smooth-
 1349 ing parameter. *Cons:* often has poor accuracy, especially with correlated features.

1350 **Perceptron** *Pros:* easy to implement; online; error-driven learning means that accuracy
 1351 is typically high, especially after averaging. *Cons:* not probabilistic; hard to know
 1352 when to stop learning; lack of margin can lead to overfitting.

1353 **Support vector machine** *Pros:* optimizes an error-based metric, usually resulting in high
 1354 accuracy; overfitting is controlled by a regularization parameter. *Cons:* not proba-
 1355 bilistic.

1356 **Logistic regression** *Pros:* error-driven and probabilistic; overfitting is controlled by a reg-
 1357 ularization parameter. *Cons:* batch learning requires black-box optimization; logistic
 1358 loss can “overtrain” on correctly labeled examples.

1359 One of the main distinctions is whether the learning algorithm offers a probability
 1360 over labels. This is useful in modular architectures, where the output of one classifier
 1361 is the input for some other system. In cases where probability is not necessary, the sup-
 1362 port vector machine is usually the right choice, since it is no more difficult to implement
 1363 than the perceptron, and is often more accurate. When probability is necessary, logistic
 1364 regression is usually more accurate than Naïve Bayes.

1365 Additional resources

1366 For more on classification, you should consult a textbook on machine learning (e.g., Mur-
 1367 phy, 2012), although the notation will differ slightly from what is typical in natural lan-
 1368 guage processing. Probabilistic methods are surveyed by Hastie et al. (2009), and Mohri
 1369 et al. (2012) emphasize theoretical considerations. Bottou et al. (2016) surveys the rapidly
 1370 moving field of online learning, and Kummerfeld et al. (2015) empirically review several

optimization algorithms for large-margin learning. The python toolkit SCIKIT-LEARN includes implementations of all of the algorithms described in this chapter (Pedregosa et al., 2011).

Appendix B describes an alternative large-margin classifier, called **passive-aggressive**. Passive-aggressive is an online learner that seeks to make the smallest update that satisfies the margin constraint at the current instance. It is closely related to MIRA, which was used widely in NLP in the 2000s (Crammer and Singer, 2003).

Exercises

There will be exercises at the end of each chapter. In this chapter, the exercises are mostly mathematical, matching the subject material. In other chapters, the exercises will emphasize linguistics or programming.

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

$$p(x | y) = \begin{cases} \alpha, & X = 0, Y = 0 \\ 1 - \alpha, & X = 1, Y = 0 \\ 1 - \beta, & X = 0, Y = 1 \\ \beta, & X = 1, Y = 1. \end{cases} \quad [2.79]$$

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

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

$$\forall y, y' \in \mathcal{Y}, \theta \cdot f(\mathbf{x}, y) - \theta \cdot f(\mathbf{x}, y') = \theta' \cdot f'(\mathbf{x}, y) - \theta' \cdot f'(\mathbf{x}, y'). \quad [2.80]$$

- 1396 b) Explain how your construction justifies the well-known alternative form for
 1397 binary logistic regression, $\Pr(Y = 1 \mid \mathbf{x}; \boldsymbol{\theta}) = \frac{1}{1+\exp(-\boldsymbol{\theta}' \cdot \mathbf{x})} = \sigma(\boldsymbol{\theta}' \cdot \mathbf{x})$, where σ
 1398 is the sigmoid function.
- 1399 5. Suppose you have two labeled datasets D_1 and D_2 , with the same features and la-
 1400 bels.
- 1401 • Let $\boldsymbol{\theta}^{(1)}$ be the unregularized logistic regression (LR) coefficients from training
 1402 on dataset D_1 .
- 1403 • Let $\boldsymbol{\theta}^{(2)}$ be the unregularized LR coefficients (same model) from training on
 1404 dataset D_2 .
- 1405 • Let $\boldsymbol{\theta}^*$ be the unregularized LR coefficients from training on the combined
 1406 dataset $D_1 \cup D_2$.

Under these conditions, prove that for any feature j ,

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

- 1407
- 1408 6. Let $\hat{\boldsymbol{\theta}}$ be the solution to an unregularized logistic regression problem, and let $\boldsymbol{\theta}^*$ be
 1409 the solution to the same problem, with L_2 regularization. Prove that $\|\boldsymbol{\theta}^*\|_2^2 \leq \|\hat{\boldsymbol{\theta}}\|_2^2$.
- 1410 7. As noted in the discussion of averaged perceptron in § 2.3.2, the computation of the
 1411 running sum $\mathbf{m} \leftarrow \mathbf{m} + \boldsymbol{\theta}$ is unnecessarily expensive, requiring $K \times V$ operations.
 1412 Give an alternative way to compute the averaged weights $\bar{\boldsymbol{\theta}}$, with complexity that is
 1413 independent of V and linear in the sum of feature sizes $\sum_{i=1}^N |\mathbf{f}(\mathbf{x}^{(i)}, y^{(i)})|$.
- 1414 8. Consider a dataset that is comprised of two identical instances $\mathbf{x}^{(1)} = \mathbf{x}^{(2)}$ with
 1415 distinct labels $y^{(1)} \neq y^{(2)}$. Assume all features are binary, $x_j \in \{0, 1\}$ for all j .

1416 Now suppose that the averaged perceptron always trains on the instance $(\mathbf{x}^{i(t)}, y^{i(t)})$,
 1417 where $i(t) = 2 - (t \bmod 2)$, which is 1 when the training iteration t is odd, and 2
 1418 when t is even. Further suppose that learning terminates under the following con-
 1419 dition:

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

1420 In words, the algorithm stops when the largest change in the averaged weights is
 1421 less than or equal to ϵ . Compute the number of iterations before the averaged per-
 1422 ceptron terminates.

- 1423 9. Prove that the margin loss is convex in θ . Use this definition of the margin loss:

$$L(\theta) = -\theta \cdot f(x, y^*) + \max_y \theta \cdot f(x, y) + c(y^*, y), \quad [2.82]$$

1424 where y^* is the gold label. As a reminder, a function f is convex iff,

$$f(\alpha x_1 + (1 - \alpha)x_2) \leq \alpha f(x_1) + (1 - \alpha)f(x_2), \quad [2.83]$$

1425 for any x_1, x_2 and $\alpha \in [0, 1]$.

- 1426 10. If a function f is m -strongly convex, then for some $m > 0$, the following inequality
1427 holds for all x and y on the domain of the function:

$$f(y) \leq f(x) + (\nabla_x f) \cdot (y - x) + \frac{m}{2} \|y - x\|_2^2. \quad [2.84]$$

1428 Let $f(x) = L(\theta^{(t)})$, the loss of the classifier at iteration t of gradient descent, and let
1429 $f(y) = L(\theta^{(t+1)})$. Assuming the loss function is m -convex, prove that $L(\theta^{(t+1)}) \leq$
1430 $L(\theta^{(t)})$ for an appropriate constant learning rate η , which will depend on m . Explain
1431 why this implies that gradient descent converges when applied to an m -strongly
1432 convex loss function with a unique minimum.

1433 **Chapter 3**

1434 **Nonlinear classification**

1435 Linear classification may seem like all we need for natural language processing. The bag-
1436 of-words representation is inherently high dimensional, and the number of features is
1437 often larger than the number of labeled training instances. This means that it is usually
1438 possible to find a linear classifier that perfectly fits the training data, or even to fit any ar-
1439bitrary labeling of the training instances! Moving to nonlinear classification may therefore
1440only increase the risk of overfitting. Furthermore, for many tasks, **lexical features** (words)
1441are meaningful in isolation, and can offer independent evidence about the instance label
1442— unlike computer vision, where individual pixels are rarely informative, and must be
1443evaluated holistically to make sense of an image. For these reasons, natural language
1444processing has historically focused on linear classification.

1445 But in recent years, nonlinear classifiers have swept through natural language pro-
1446cessing, and are now the default approach for many tasks (Manning, 2015). There are at
1447least three reasons for this change.

- 1448 • There have been rapid advances in **deep learning**, a family of nonlinear meth-
1449ods that learn complex functions of the input through multiple layers of computa-
1450tion (Goodfellow et al., 2016).
- 1451 • Deep learning facilitates the incorporation of **word embeddings**, which are dense
1452vector representations of words. Word embeddings can be learned from large amounts
1453of unlabeled data, and enable generalization to words that do not appear in the an-
1454notated training data (word embeddings are discussed in detail in chapter 14).
- 1455 • While CPU speeds have plateaued, there have been rapid advances in specialized
1456hardware called graphics processing units (GPUs), which have become faster, cheaper,
1457and easier to program. Many deep learning models can be implemented efficiently
1458on GPUs, offering substantial performance improvements over CPU-based comput-
1459ing.

1460 This chapter focuses on **neural networks**, which are the dominant approach for non-
 1461 linear classification in natural language processing today.¹ Historically, a few other non-
 1462 linear learning methods have been applied to language data.

- 1463 • **Kernel methods** are generalizations of the **nearest-neighbor** classification rule, which
 1464 classifies each instance by the label of the most similar example in the training set.
 1465 The application of the **kernel support vector machine** to information extraction is
 1466 described in chapter 17.
- 1467 • **Decision trees** classify instances by checking a set of conditions. Scaling decision
 1468 trees to bag-of-words inputs is difficult, but decision trees have been successful in
 1469 problems such as coreference resolution (chapter 15), where more compact feature
 1470 sets can be constructed (Soon et al., 2001).
- 1471 • **Boosting** and related **ensemble methods** work by combining the predictions of sev-
 1472 eral “weak” classifiers, each of which may consider only a small subset of features.
 1473 Boosting has been successfully applied to text classification (Schapire and Singer,
 1474 2000) and syntactic analysis (Abney et al., 1999), and remains one of the most suc-
 1475 cessful methods on machine learning competition sites such as Kaggle (Chen and
 1476 Guestrin, 2016).

1477 Hastie et al. (2009) provide an excellent overview of these techniques.

1478 3.1 Feedforward neural networks

1479 Consider the problem of building a classifier for movie reviews. The goal is to predict a
 1480 label $y \in \{\text{GOOD}, \text{BAD}, \text{OKAY}\}$ from a representation of the text of each document, x . But
 1481 what makes a good movie? The story, acting, cinematography, editing, soundtrack, and
 1482 so on. Now suppose the training set contains labels for each of these additional features,
 1483 $z = [z_1, z_2, \dots, z_{K_z}]^\top$. With a training set of such information, we could build a two-step
 1484 classifier:

- 1485 1. **Use the text x to predict the features z .** Specifically, train a logistic regression clas-
 1486 sifier to compute $p(z_k | x)$, for each $k \in \{1, 2, \dots, K_z\}$.
- 1487 2. **Use the features z to predict the label y .** Again, train a logistic regression classifier
 1488 to compute $p(y | z)$. On test data, z is unknown, so we will use the probabilities
 1489 $p(z | x)$ from the first layer as the features.

1490 This setup is shown in Figure 3.1, which describes the proposed classifier in a **computation**
 1491 **graph**: the text features x are connected to the middle layer z , which is connected to
 1492 the label y .

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

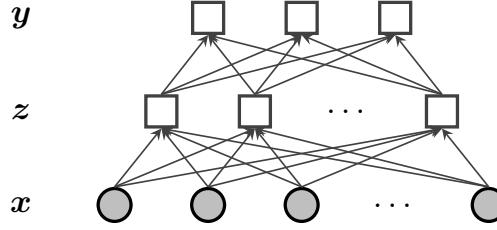


Figure 3.1: A feedforward neural network. Shaded circles indicate observed features, usually words; squares indicate nodes in the computation graph, which are computed from the information carried over the incoming arrows.

1493 If we assume that each z_k is binary, $z_k \in \{0, 1\}$, then the probability $p(z_k | x)$ can be
1494 modeled using binary logistic regression:

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

1495 where σ is the **sigmoid** function (shown in Figure 3.2), and the matrix $\Theta^{(x \rightarrow z)} \in \mathbb{R}^{K_z \times V}$ is
1496 constructed by stacking the weight vectors for each z_k ,

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

1497 We will assume that x contains a term with a constant value of 1, so that a corresponding
1498 offset parameter is included in each $\theta_k^{(x \rightarrow z)}$.

1499 The output layer is computed by the multi-class logistic regression probability,

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

1500 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
1501 constructed by concatenation,

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

1502 The vector of probabilities over each possible value of y is denoted,

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

1503 where element j in the output of the **SoftMax** function is computed as in Equation 3.3.

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

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

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

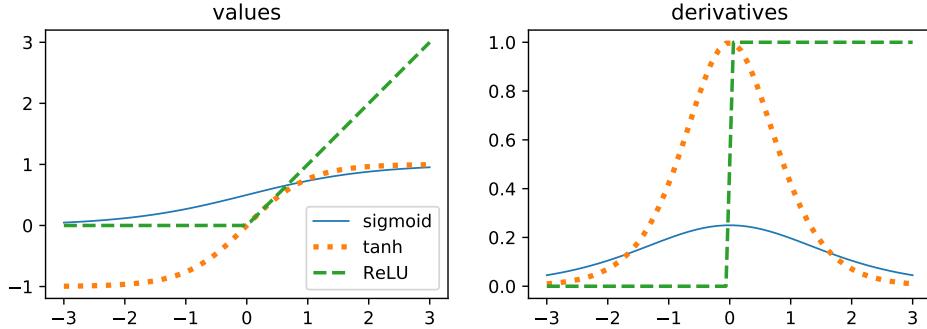


Figure 3.2: The sigmoid, tanh, and ReLU activation functions

1504 where the function σ is now applied **elementwise** to the vector of inner products,

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

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

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

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

1505 This defines a classification model that predicts the label $y \in \mathcal{Y}$ from the base features \mathbf{x} ,
 1506 through a “hidden layer” \mathbf{z} . This is a **feedforward neural network**.²

1507 3.2 Designing neural networks

1508 There several ways to generalize the feedforward neural network.

1509 3.2.1 Activation functions

1510 If the hidden layer is viewed as a set of latent features, then the sigmoid function repre-
 1511 sents the extent to which each of these features is “activated” by a given input. However,
 1512 the hidden layer can be regarded more generally as a nonlinear transformation of the in-
 1513 put. This opens the door to many other activation functions, some of which are shown in
 1514 Figure 3.2. At the moment, the choice of activation functions is more art than science, but
 1515 a few points can be made about the most popular varieties:

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

- The range of the sigmoid function is $(0, 1)$. The bounded range ensures that a cascade of sigmoid functions will not “blow up” to a huge output, and this is important for deep networks with several hidden layers. The derivative of the sigmoid is $\frac{\partial}{\partial a}\sigma(a) = \sigma(a)(1 - \sigma(a))$. This derivative becomes small at the extremes, which can make learning slow; this is called the **vanishing gradient** problem.
- The range of the **tanh activation function** is $(-1, 1)$: like the sigmoid, the range is bounded, but unlike the sigmoid, it includes negative values. The derivative is $\frac{\partial}{\partial a}\tanh(a) = 1 - \tanh(a)^2$, which is steeper than the logistic function near the origin (LeCun et al., 1998). The tanh function can also suffer from vanishing gradients at extreme values.
- The **rectified linear unit (ReLU)** is zero for negative inputs, and linear for positive inputs (Glorot et al., 2011),

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

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

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

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

3.2.2 Network structure

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

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

1545 It is also possible to “short circuit” a hidden layer, by propagating information directly
 1546 from the input to the next higher level of the network. This is the idea behind **residual net-**
 1547 **works**, which propagate information directly from the input to the subsequent layer (He
 1548 et al., 2016),

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

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

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

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

1549 where \odot refers to an elementwise vector product, and 1 is a column vector of ones. As
 1550 before, the sigmoid function is applied elementwise to its input; recall that the output of
 1551 this function is restricted to the range $(0, 1)$. Gating is also used in the **long short-term**
 1552 **memory (LSTM)**, which is discussed in chapter 6. Residual and highway connections
 1553 address a problem with deep architectures: repeated application of a nonlinear activation
 1554 function can make it difficult to learn the parameters of the lower levels of the network,
 1555 which are too distant from the supervision signal.

1556 3.2.3 Outputs and loss functions

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

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

1557 where $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, b\}$ is the entire set of parameters.

This loss can be written alternatively as follows:

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

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

for a survey of these theoretical results. However, depending on the function to be approximated, the width of the hidden layer may need to be arbitrarily large. Furthermore, the fact that a network has the *capacity* to approximate any given function does not imply that it is possible to *learn* the function using gradient-based optimization.

1558 where $e_{y^{(i)}}$ is a **one-hot vector** of zeros with a value of 1 at position $y^{(i)}$. The inner product
 1559 between $e_{y^{(i)}}$ and \tilde{y} is also called the multinomial **cross-entropy**, and this terminology
 1560 is preferred in many neural networks papers and software packages.

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

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

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

1561 In regression problems, the output is a scalar or vector (see § 4.1.2). For these problems, a
 1562 typical loss function is the squared error $(y - \hat{y})^2$ or squared norm $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$.

1563 3.2.4 Inputs and lookup layers

1564 In text classification, the input layer \mathbf{x} can refer to a bag-of-words vector, where x_j is
 1565 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
 1566 represented by the vector $\theta_j^{(x \rightarrow z)}$. This vector is sometimes described as the **embedding** of
 1567 word j , and can be learned from unlabeled data, using techniques discussed in chapter 14.
 1568 The columns of $\Theta^{(x \rightarrow z)}$ are each K_z -dimensional word embeddings.

1569 Chapter 2 presented an alternative view of text documents, as a sequence of word
 1570 tokens, w_1, w_2, \dots, w_M . In a neural network, each word token w_m is represented with a
 1571 one-hot vector, e_{w_m} , with dimension V . The matrix-vector product $\Theta^{(x \rightarrow z)} e_{w_m}$ returns
 1572 the embedding of word w_m . The complete document can be represented by horizontally
 1573 concatenating these one-hot vectors, $\mathbf{W} = [e_{w_1}, e_{w_2}, \dots, e_{w_M}]$, and the bag-of-words rep-
 1574 resentation can be recovered from the matrix-vector product $\mathbf{W}[1, 1, \dots, 1]^\top$, which sums
 1575 each row over the tokens $m = \{1, 2, \dots, M\}$. The matrix product $\Theta^{(x \rightarrow z)} \mathbf{W}$ contains the
 1576 horizontally concatenated embeddings of each word in the document, which will be use-
 1577 ful as the starting point for **convolutional neural networks** (see § 3.4). This is sometimes
 1578 called a **lookup layer**, because the first step is to lookup the embeddings for each word in
 1579 the input text.

1580 3.3 Learning neural networks

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

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

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

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

where f is an elementwise activation function, such as σ or ReLU, and $\ell^{(i)}$ is the loss at instance i . The parameters $\Theta^{(x \rightarrow z)}$, $\Theta^{(z \rightarrow y)}$, and b can be estimated using online gradient-based optimization. The simplest such algorithm is stochastic gradient descent, which was discussed in § 2.6. Each parameter is updated by the gradient of the loss,

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

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

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

1581 where $\eta^{(t)}$ is the learning rate on iteration t , $\ell^{(i)}$ is the loss on instance (or minibatch) i ,
1582 and $\boldsymbol{\theta}_n^{(x \rightarrow z)}$ is column n of the matrix $\Theta^{(x \rightarrow z)}$, and $\boldsymbol{\theta}_k^{(z \rightarrow y)}$ is column k of $\Theta^{(z \rightarrow y)}$.

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

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

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

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

1583 where $\delta(j = y^{(i)})$ is a function that returns one when $j = y^{(i)}$, and zero otherwise. The
1584 gradient $\nabla_{\mathbf{b}} \ell^{(i)}$ is similar to Equation 3.29.

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

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

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

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

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

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

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

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

1585 For intuition, consider each of the terms in the product.

- 1586 • If the negative log-likelihood $\ell^{(i)}$ does not depend much on z_k , then $\frac{\partial \ell^{(i)}}{\partial z_k} \approx 0$. In this
1587 case it doesn't matter how z_k is computed, and so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} \approx 0$.
- 1588 • If z_k is near 1 or 0, then the curve of the sigmoid function is nearly flat (Figure 3.2),
1589 and changing the inputs will make little local difference. The term $z_k \times (1 - z_k)$ is
1590 maximized at $z_k = \frac{1}{2}$, where the slope of the sigmoid function is steepest.
- 1591 • If $x_n = 0$, then it does not matter how we set the weights $\theta_{n,k}^{(x \rightarrow z)}$, so $\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \rightarrow z)}} = 0$.

1592 3.3.1 Backpropagation

1593 The equations above rely on the chain rule to compute derivatives of the loss with respect
1594 to each parameter of the model. Furthermore, local derivatives are frequently reused: for
1595 example, $\frac{\partial \ell^{(i)}}{\partial z_k}$ is reused in computing the derivatives with respect to each $\theta_{n,k}^{(x \rightarrow z)}$. These
1596 terms should therefore be computed once, and then cached. Furthermore, we should only
1597 compute any derivative once we have already computed all of the necessary "inputs"
1598 demanded by the chain rule of differentiation. This combination of sequencing, caching,
1599 and differentiation is known as **backpropagation**. It can be generalized to any directed
1600 acyclic **computation graph**.

1601 A computation graph is a declarative representation of a computational process. At
1602 each node t , compute a value v_t by applying a function f_t to a (possibly empty) list of
1603 parent nodes, π_t . For example, in a feedforward network with one hidden layer, there are
1604 nodes for the input $\mathbf{x}^{(i)}$, the hidden layer \mathbf{z} , the predicted output $\tilde{\mathbf{y}}$, and the parameters
1605 $\{\Theta^{(x \rightarrow z)}, \Theta^{(z \rightarrow y)}, \mathbf{b}\}$. During training, there is also a node for the label $\mathbf{y}^{(i)}$ and the loss
1606 $\ell^{(i)}$. The predicted output $\tilde{\mathbf{y}}$ is one of the parents of the loss (the other is the label $\mathbf{y}^{(i)}$); its
1607 parents include $\Theta^{(z \rightarrow y)}$ and \mathbf{z} , and so on.

1608 Computation graphs include three types of nodes:

1609 **Variables.** The variables include the inputs \mathbf{x} , the hidden nodes \mathbf{z} , the outputs \mathbf{y} , and
1610 the loss function. Inputs are variables that do not have parents. Backpropagation

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

```

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  $\{g_1, g_2, \dots, g_T\}$ 

```

1611 computes the gradients with respect to all variables except the inputs, but does not
 1612 update the variables during learning.

1613 **Parameters.** In a feedforward network, the parameters include the weights and offsets.
 1614 Parameter nodes do not have parents, and they are updated during learning.

1615 **Objective.** The *objective* node is not the parent of any other node. Backpropagation begins
 1616 by computing the gradient with respect to this node.

1617 If the computation graph is a directed acyclic graph, then it is possible to order the
 1618 nodes with a topological sort, so that if node t is a parent of node t' , then $t < t'$. This
 1619 means that the values $\{v_t\}_{t=1}^T$ can be computed in a single forward pass. The topolog-
 1620 ical sort is reversed when computing gradients: each gradient g_t is computed from the
 1621 gradients of the children of t , implementing the chain rule of differentiation. The general
 1622 backpropagation algorithm for computation graphs is shown in Algorithm 6, and illus-
 1623 trated in Figure 3.3.

1624 While the gradients with respect to each parameter may be complex, they are com-
 1625 posed of products of simple parts. For many networks, all gradients can be computed
 1626 through **automatic differentiation**. This means that end users need only specify the feed-
 1627 forward computation, and the gradients necessary for learning can be obtained automati-
 1628 cally. There are many software libraries that perform automatic differentiation on compu-
 1629 tation graphs, such as TORCH (Collobert et al., 2011), TENSORFLOW (Abadi et al., 2016),
 1630 and DYNET (Neubig et al., 2017). One important distinction between these libraries is
 1631 whether they support **dynamic computation graphs**, in which the structure of the compu-
 1632 tation graph varies across instances. Static computation graphs are compiled in advance,
 1633 and can be applied to fixed-dimensional data, such as bag-of-words vectors. In many nat-

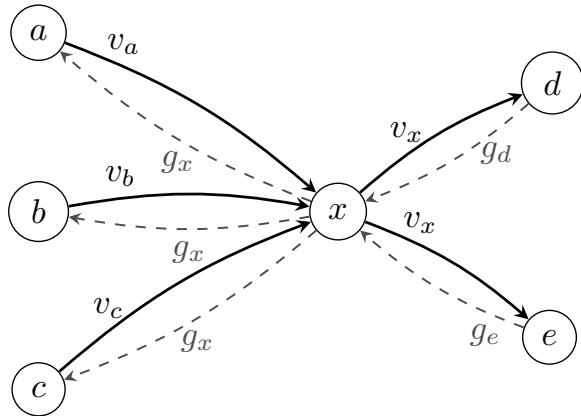


Figure 3.3: Backpropagation at a single node x in the computation graph. The values of the predecessors v_a, v_b, v_c are the inputs to x , which computes v_x , and passes it on to the successors d and e . The gradients at the successors g_d and g_e are passed back to x , where they are incorporated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

1634 ural language processing problems, each input has a distinct structure, requiring a unique
 1635 computation graph.

1636 3.3.2 Regularization and dropout

1637 In linear classification, overfitting was addressed by augmenting the objective with a reg-
 1638 ularization term, $\lambda \|\boldsymbol{\theta}\|_2^2$. This same approach can be applied to feedforward neural net-
 1639 works, penalizing each matrix of weights:

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

1640 where $\|\boldsymbol{\Theta}\|_F^2 = \sum_{i,j} \theta_{i,j}^2$ is the squared **Frobenius norm**, which generalizes the L_2 norm
 1641 to matrices. The bias parameters b are not regularized, as they do not contribute to the
 1642 sensitivity of the classifier to the inputs. In gradient-based optimization, the practical
 1643 effect of Frobenius norm regularization is that the weights “decay” towards zero at each
 1644 update, motivating the alternative name **weight decay**.

1645 Another approach to controlling model complexity is **dropout**, which involves ran-
 1646 domly setting some computation nodes to zero during training (Srivastava et al., 2014).
 1647 For example, in the feedforward network, on each training instance, with probability ρ we
 1648 set each input x_n and each hidden layer node z_k to zero. Srivastava et al. (2014) recom-
 1649 mend $\rho = 0.5$ for hidden units, and $\rho = 0.2$ for input units. Dropout is also incorporated

in the gradient computation, so if node z_k is dropped, then none of the weights $\theta_k^{(x \rightarrow z)}$ will be updated for this instance. Dropout prevents the network from learning to depend too much on any one feature or hidden node, and prevents **feature co-adaptation**, in which a hidden unit is only useful in combination with one or more other hidden units. Dropout is a special case of **feature noising**, which can also involve adding Gaussian noise to inputs or hidden units (Holmstrom and Koistinen, 1992). Wager et al. (2013) show that dropout is approximately equivalent to “adaptive” L_2 regularization, with a separate regularization penalty for each feature.

3.3.3 *Learning theory

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

The non-convexity of hidden layer neural networks can also be seen by permuting the elements of the hidden layer, from $z = [z_1, z_2, \dots, z_{K_z}]$ to $\tilde{z} = [\tilde{z}_{\pi(1)}, \tilde{z}_{\pi(2)}, \dots, \tilde{z}_{\pi(K_z)}]$. This corresponds to applying π to the rows of $\Theta^{(x \rightarrow z)}$ and the columns of $\Theta^{(z \rightarrow y)}$, resulting in permuted parameter matrices $\Theta_\pi^{(x \rightarrow z)}$ and $\Theta_\pi^{(z \rightarrow y)}$. As long as this permutation is applied consistently, the loss will be identical, $L(\Theta) = L(\Theta_\pi)$: it is *invariant* to this permutation. However, the loss of the linear combination $L(\alpha\Theta + (1 - \alpha)\Theta_\pi)$ will generally not be identical to the loss under Θ or its permutations. If $L(\Theta)$ is better than the loss at any points in the immediate vicinity, and if $L(\Theta) = L(\Theta_\pi)$, then the loss function does not satisfy the definition of convexity (see § 2.4). 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). More generally, a **critical point** is one at which the gradient is zero. Critical points may be local optima, but they may also be **saddle points**, which are local minima in some directions, but local *maxima* in other directions. For example, the equation $x_1^2 - x_2^2$ has a saddle point at $x = (0, 0)$. In large networks, the overwhelming majority of critical points are saddle points, rather than local minima or maxima (Dauphin et al., 2014). Saddle points can pose problems for gradient-based optimization, since learning will slow to a crawl as the gradient goes

1688 to zero. However, the noise introduced by stochastic gradient descent, and by feature
 1689 noising techniques such as dropout, can help online optimization to escape saddle points
 1690 and find high-quality optima (Ge et al., 2015). Other techniques address saddle points
 1691 directly, using local reconstructions of the Hessian matrix (Dauphin et al., 2014) or higher-
 1692 order derivatives (Anandkumar and Ge, 2016).

1693 **3.3.4 Tricks**

1694 Getting neural networks to work effectively sometimes requires heuristic “tricks” (Bottou,
 1695 2012; Goodfellow et al., 2016; Goldberg, 2017b). This section presents some tricks that are
 1696 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]

1697 For the weights leading to a ReLU activation function, He et al. (2015) use similar argu-
 1698 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:

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

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

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

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

1699 The matrix \mathbf{U} contains the **singular vectors** of \mathbf{A} , and is guaranteed to be orthonormal.
1700 For more on singular value decomposition, see chapter 14.

1701 Even with careful initialization, there can still be significant variance in the final re-
1702 sults. It can be useful to make multiple training runs, and select the one with the best
1703 performance on a heldout development set.

1704 **Clipping and normalizing the gradients** As already discussed, the magnitude of the
1705 gradient can pose problems for learning: too large, and learning can diverge, with suc-
1706 ccessive updates thrashing between increasingly extreme values; too small, and learning can
1707 grind to a halt. Several heuristics have been proposed to address this issue.

1708 • In **gradient clipping** (Pascanu et al., 2013), an upper limit is placed on the norm of
1709 the gradient, and the gradient is rescaled when this limit is exceeded,

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

1708 • In **batch normalization** (Ioffe and Szegedy, 2015), the inputs to each computation
1709 node are recentered by their mean and variance across all of the instances in the
minibatch \mathcal{B} (see § 2.6.2). For example, in a feedforward network with one hidden
layer, batch normalization would transform the inputs to the hidden layer as follows:

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

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

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

1710 Empirically, this speeds convergence of deep architectures. One explanation is that
1711 it helps to correct for changes in the distribution of activations during training.

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

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

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

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

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

1712 Layer normalization has similar motivations to batch normalization, but it can be
 1713 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 § 2.6.2; 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,

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

1714 where $g_{t,j}$ is the gradient with respect to parameter j at time t , and $\beta \in [0, 1]$. This term
 1715 places more emphasis on recent gradients, and is employed in the AdaDelta (Zeiler, 2012)
 1716 and Adam (Kingma and Ba, 2014) optimizers. Online optimization and its theoretical
 1717 background are reviewed by Bottou et al. (2016). **Early stopping**, mentioned in § 2.3.2,
 1718 can help to avoid overfitting, by terminating training after reaching a plateau in the per-
 1719 formance on a heldout validation set.

1720 3.4 Convolutional neural networks

1721 A basic weakness of the bag-of-words model is its inability to account for the ways in
 1722 which words combine to create meaning, including even simple reversals such as *not*
 1723 *pleasant*, *hardly a generous offer*, and *I wouldn't mind missing the flight*. Computer vision
 1724 faces the related challenge of identifying the semantics of images from pixel features
 1725 that are uninformative in isolation. An earlier generation of computer vision research
 1726 focused on designing *filters* to aggregate local pixel-level features into more meaningful
 1727 representations, such as edges and corners (e.g., Canny, 1987). Similarly, earlier NLP re-
 1728 search attempted to capture multiword linguistic phenomena by hand-designed lexical
 1729 patterns (Hobbs et al., 1997). In both cases, the output of the filters and patterns could

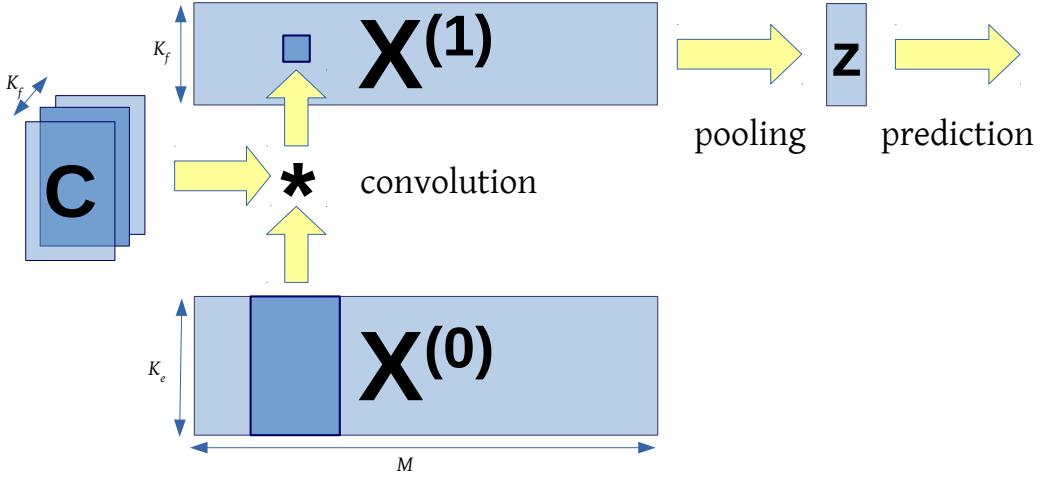


Figure 3.4: A convolutional neural network for text classification

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

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

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

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

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

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

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

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

1738 thereby representing trigram units like *not so unpleasant*. In **one-dimensional convolution**,
 1739 each filter matrix $\mathbf{C}^{(k)}$ is constrained to have non-zero values only at row k (Kalchbrenner et al., 2014). This means that each dimension of the word embedding is processed
 1740 by a separate filter, and it implies that $K_f = K_e$.
 1741

1742 To deal with the beginning and end of the input, the base matrix $\mathbf{X}^{(0)}$ may be padded
 1743 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
 1744 than the input; this is known as **narrow convolution**. The filter matrices need not have
 1745 identical filter widths, so more generally we could write h_k to indicate width of filter
 1746 $\mathbf{C}^{(k)}$. As suggested by the notation $\mathbf{X}^{(0)}$, multiple layers of convolution may be applied,
 1747 so that $\mathbf{X}^{(d)}$ is the input to $\mathbf{X}^{(d+1)}$.
 1748

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,

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

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

1749 The vector \mathbf{z} can now act as a layer in a feedforward network, culminating in a prediction
 1750 \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 :

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

1751 The computer vision literature has produced a huge variety of convolutional architectures, and many of these innovations can be applied to text data. One avenue for
 1752 improvement is more complex pooling operations, such as k -max pooling (Kalchbrenner
 1753 et al., 2014), which returns a matrix of the k largest values for each filter. Another innovation
 1754 is the use of **dilated convolution** to build multiscale representations (Yu and Koltun,
 1755 2016). At each layer, the convolutional operator applied in *strides*, skipping ahead by s
 1756 steps after each feature. As we move up the hierarchy, each layer is s times smaller than
 1757 the layer below it, effectively summarizing the input (Kalchbrenner et al., 2016; Strubell
 1758 et al., 2017). This idea is shown in Figure 3.5. Multi-layer convolutional networks can also
 1759

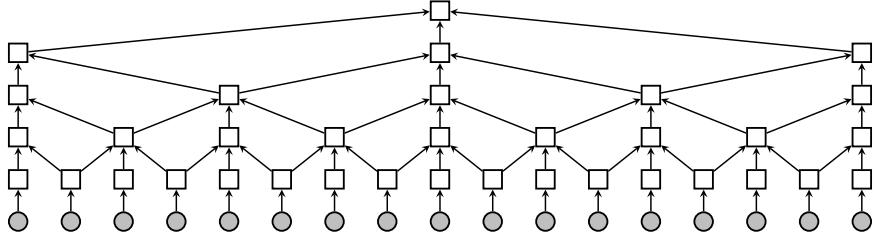


Figure 3.5: A dilated convolutional neural network captures progressively larger context through recursive application of the convolutional operator

1760 be augmented with “shortcut” connections, as in the residual network from § 3.2.2 (Johnson and Zhang, 2017).
 1761

1762 Additional resources

1763 The deep learning textbook by Goodfellow et al. (2016) covers many of the topics in this
 1764 chapter in more detail. For a comprehensive review of neural networks in natural lan-
 1765 guage processing, see Goldberg (2017b). A seminal work on deep learning in natural
 1766 language processing is the aggressively titled “Natural Language Processing (Almost)
 1767 from Scratch”, which uses convolutional neural networks to perform a range of language
 1768 processing tasks (Collobert et al., 2011), although there is earlier work (e.g., Henderson,
 1769 2004). This chapter focuses on feedforward and convolutional neural networks, but recur-
 1770 rent neural networks are one of the most important deep learning architectures for natural
 1771 language processing. They are covered extensively in chapters 6 and 7.

1772 The role of deep learning in natural language processing research has caused angst
 1773 in some parts of the natural language processing research community (e.g., Goldberg,
 1774 2017a), especially as some of the more zealous deep learning advocates have argued that
 1775 end-to-end learning from “raw” text can eliminate the need for linguistic constructs such
 1776 as sentences, phrases, and even words (Zhang et al., 2015, originally titled *Text understand-
 1777 ing from scratch*). These developments were surveyed by Manning (2015). While reports of
 1778 the demise of linguistics in natural language processing remain controversial at best, deep
 1779 learning and backpropagation have become ubiquitous in both research and applications.

1780 Exercises

- 1781 1. a) Draw the computation graph for a feedforward network with a single hidden
 1782 layer. You may represent the vector of values at each layer as a single node.
 1783 Don’t forget to include the parameters, the label, and the loss.
 1784 b) Update your computation graph to include a residual connection.

- 1785 c) Update your computation graph to include a highway connection.
- 1786 2. Prove that the softmax and sigmoid functions are equivalent when the number of
 1787 possible labels is two. Specifically, for any $\Theta^{(z \rightarrow y)}$ (omitting the offset b for simplic-
 1788 ity), show how to construct a vector of weights θ such that,

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

- 1789 3. Convolutional neural networks often aggregate across words by using **max-pooling**
 1790 (Equation 3.55 in § 3.4). A potential concern is that there is zero gradient with re-
 1791 spect to the parts of the input that are not included in the maximum. The following
 1792 questions consider the gradient with respect to an element of the input, $x_{m,k}^{(0)}$, and
 1793 they assume that all parameters are independently distributed.

- 1794 a) First consider a minimal network, with $z = \text{MaxPool}(\mathbf{X}^{(0)})$. What is the prob-
 1795 ability that the gradient $\frac{\partial \ell}{\partial x_{m,k}^{(0)}}$ is non-zero?
- 1796 b) Now consider a two-level network, with $\mathbf{X}^{(1)} = f(\mathbf{b} + \mathbf{C} * \mathbf{X}^{(0)})$. Express the
 1797 probability that the gradient $\frac{\partial \ell}{\partial x_{m,k}^{(0)}}$ is non-zero, in terms of the input length M ,
 1798 the filter size n , and the number of filters K_f .
- 1799 c) Using a calculator, work out the probability for the case $M = 128, n = 4, K_f =$
 1800 32.
- 1801 d) Now consider a three-level network, $\mathbf{X}^{(2)} = f(\mathbf{b} + \mathbf{C} * \mathbf{X}^{(1)})$. Give the general
 1802 equation for the probability that $\frac{\partial \ell}{\partial x_{m,k}^{(0)}}$ is non-zero, and compute the numerical
 1803 probability for the scenario in the previous part, assuming $K_f = 32$ and $n = 4$
 1804 at both levels.

- 1805 4. Design a feedforward network to compute the XOR function:

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

1806 Your network should have a single output node which uses the Sign activation func-
 1807 tion, $f(x) = \begin{cases} 1, & x > 0 \\ -1, & x \leq 0 \end{cases}$. Use a single hidden layer, with ReLU activation func-
 1808 tions. Describe all weights and offsets.

- 1809 5. Consider the same network as above (with ReLU activations for the hidden layer),
 1810 with an arbitrary differentiable loss function $\ell(y^{(i)}, \tilde{y})$, where \tilde{y} is the activation of
 1811 the output node. Suppose all weights and offsets are initialized to zero. Show that
 1812 gradient descent will not learn the desired function from this initialization.
- 1813 6. The simplest solution to the previous problem relies on the use of the ReLU activa-
 1814 tion function at the hidden layer. Now consider a network with arbitrary activations
 1815 on the hidden layer. Show that if the initial weights are any uniform constant, then
 1816 gradient descent will not learn the desired function from this initialization.
- 1817 7. Consider a network in which: the base features are all binary, $\mathbf{x} \in \{0, 1\}^M$; the
 1818 hidden layer activation function is sigmoid, $z_k = \sigma(\theta_k \cdot \mathbf{x})$; and the initial weights
 1819 are sampled independently from a standard normal distribution, $\theta_{j,k} \sim N(0, 1)$.
- 1820 • Show how the probability of a small initial gradient on any weight, $\frac{\partial z_k}{\partial \theta_{j,k}} < \alpha$,
 1821 depends on the size of the input M . **Hint:** use the lower bound,
- $$\Pr(\sigma(\theta_k \cdot \mathbf{x}) \times (1 - \sigma(\theta_k \cdot \mathbf{x})) < \alpha) \geq 2 \Pr(\sigma(\theta_k \cdot \mathbf{x}) < \alpha), \quad [3.60]$$
- 1822 and relate this probability to the variance $V[\theta_k \cdot \mathbf{x}]$.
- 1823 • Design an alternative initialization that removes this dependence.
- 1824 8. The ReLU activation function can lead to “dead neurons”, which can never be acti-
 1825 vated on any input. Consider the following two-layer feedforward network with a
 1826 scalar output y :

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

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

- 1824 Suppose that the input is a binary vector of observations, $\mathbf{x} \in \{0, 1\}^D$.
- 1825 a) Under what condition is node z_i “dead”? Your answer should be expressed in
 1826 terms of the parameters $\theta_i^{(x \rightarrow z)}$ and b_i .
- 1827 b) Suppose that the gradient of the loss on a given instance is $\frac{\partial \ell}{\partial y} = 1$. Derive the
 1828 gradients $\frac{\partial \ell}{\partial b_i}$ and $\frac{\partial \ell}{\partial \theta_{j,i}^{(x \rightarrow z)}}$ for such an instance.
- 1829 c) Using your answers to the previous two parts, explain why a dead neuron can
 1830 never be brought back to life during gradient-based learning.
- 1827 9. Suppose that the parameters $\Theta = \{\Theta^{(x \rightarrow z)}, \Theta(z \rightarrow y), \mathbf{b}\}$ are a local optimum of a
 1828 feedforward network in the following sense: there exists some $\epsilon > 0$ such that,

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

1831 Define the function π as a permutation on the hidden units, as described in § 3.3.3,
1832 so that for any Θ , $L(\Theta) = L(\Theta_\pi)$. Prove that if a feedforward network has a local
1833 optimum in the sense of Equation 3.63, then its loss is not a convex function of the
1834 parameters Θ , using the definition of convexity from § 2.4

1835 10. Consider a network with a single hidden layer, and a single output,

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

1836 Assume that g is the ReLU function. Show that for any matrix of weights $\Theta^{(x \rightarrow z)}$, it
1837 is permissible to rescale each row to have a norm of one, because an identical output
1838 can be obtained by finding a corresponding rescaling of $\theta^{(z \rightarrow y)}$.

¹⁸³⁹ Chapter 4

¹⁸⁴⁰ Linguistic applications of ¹⁸⁴¹ classification

¹⁸⁴² 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.

¹⁸⁴⁵ 4.1 Sentiment and opinion analysis

¹⁸⁴⁶ A popular application of text classification is to automatically determine the **sentiment** or **opinion polarity** of documents such as product reviews and social media posts. For example, marketers are interested to know how people respond to advertisements, services, and products (Hu and Liu, 2004); social scientists are interested in how emotions are affected by phenomena such as the weather (Hannak et al., 2012), and how both opinions and emotions spread over social networks (Coviello et al., 2014; Miller et al., 2011). In the field of **digital humanities**, literary scholars track plot structures through the flow of sentiment across a novel (Jockers, 2015).¹

¹⁸⁵⁴ Sentiment analysis can be framed as a direct application of document classification, assuming reliable labels can be obtained. In the simplest case, sentiment analysis is a two or three-class problem, with sentiments of POSITIVE, NEGATIVE, and possibly NEUTRAL. Such annotations could be annotated by hand, or obtained automatically through a variety of means:

- ¹⁸⁵⁹ • Tweets containing happy emoticons can be marked as positive, sad emoticons as negative (Read, 2005; Pak and Paroubek, 2010).

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

- 1861 • Reviews with four or more stars can be marked as positive, three or fewer stars as
 1862 negative (Pang et al., 2002).
- 1863 • Statements from politicians who are voting for a given bill are marked as positive
 1864 (towards that bill); statements from politicians voting against the bill are marked as
 1865 negative (Thomas et al., 2006).

1866 The bag-of-words model is a good fit for sentiment analysis at the document level: if
 1867 the document is long enough, we would expect the words associated with its true senti-
 1868 ment to overwhelm the others. Indeed, **lexicon-based sentiment analysis** avoids machine
 1869 learning altogether, and classifies documents by counting words against positive and neg-
 1870 ative sentiment word lists (Taboada et al., 2011).

1871 Lexicon-based classification is less effective for short documents, such as single-sentence
 1872 reviews or social media posts. In these documents, linguistic issues like **negation** and **ir-**
 1873 **realis** (Polanyi and Zaenen, 2006) — events that are hypothetical or otherwise non-factual
 1874 — can make bag-of-words classification ineffective. Consider the following examples:

- 1875 (4.1) That's not bad for the first day.
- 1876 (4.2) This is not the worst thing that can happen.
- 1877 (4.3) It would be nice if you acted like you understood.
- 1878 (4.4) There is no reason at all to believe that the polluters are suddenly going to be-
 1879 come reasonable. (Wilson et al., 2005)
- 1880 (4.5) This film should be brilliant. The actors are first grade. Stallone plays a happy,
 1881 wonderful man. His sweet wife is beautiful and adores him. He has a fascinat-
 1882 ing gift for living life fully. It sounds like a great plot, **however**, the film is a
 1883 failure. (Pang et al., 2002)

1884 A minimal solution is to move from a bag-of-words model to a bag-of-**bigrams** model,
 1885 where each base feature is a pair of adjacent words, e.g.,

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

1886 Bigrams can handle relatively straightforward cases, such as when an adjective is immedi-
 1887 ately negated; trigrams would be required to extend to larger contexts (e.g., *not the worst*).
 1888 But this approach will not scale to more complex examples like (4.4) and (4.5). More
 1889 sophisticated solutions try to account for the syntactic structure of the sentence (Wilson
 1890 et al., 2005; Socher et al., 2013), or apply more complex classifiers such as convolutional
 1891 neural networks (Kim, 2014), which are described in chapter 3.

4.1.1 Related problems

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

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

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

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

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

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

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

1928 **Emotion classification** While sentiment analysis is framed in terms of positive and neg-
 1929 ative categories, psychologists generally regard **emotion** as more multifaceted. For ex-
 1930 ample, Ekman (1992) argues that there are six basic emotions — happiness, surprise, fear,
 1931 sadness, anger, and contempt — and that they are universal across human cultures. Alm
 1932 et al. (2005) build a linear classifier for recognizing the emotions expressed in children’s
 1933 stories. The ultimate goal of this work was to improve text-to-speech synthesis, so that
 1934 stories could be read with intonation that reflected the emotional content. They used bag-
 1935 of-words features, as well as features capturing the story type (e.g., jokes, folktales), and
 1936 structural features that reflect the position of each sentence in the story. The task is diffi-
 1937 cult: even human annotators frequently disagreed with each other, and the best classifiers
 1938 achieved accuracy between 60-70%.

1939 4.1.2 Alternative approaches to sentiment analysis

1940 **Regression** A more challenging version of sentiment analysis is to determine not just
 1941 the class of a document, but its rating on a numerical scale (Pang and Lee, 2005). If the
 1942 scale is continuous, it is most natural to apply **regression**, identifying a set of weights θ
 1943 that minimize the squared error of a predictor $\hat{y} = \theta \cdot x + b$, where b is an offset. This
 1944 approach is called **linear regression**, and sometimes **least squares**, because the regression
 1945 coefficients θ are determined by minimizing the squared error, $(y - \hat{y})^2$. If the weights are
 1946 regularized using a penalty $\lambda \|\theta\|_2^2$, then it is **ridge regression**. Unlike logistic regression,
 1947 both linear regression and ridge regression can be solved in closed form as a system of
 1948 linear equations.

1949 **Ordinal ranking** In many problems, the labels are ordered but discrete: for example,
 1950 product reviews are often integers on a scale of 1 – 5, and grades are on a scale of A – F.
 1951 Such problems can be solved by discretizing the score $\theta \cdot x$ into “ranks”,

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

1952 where $b = [b_1 = -\infty, b_2, b_3, \dots, b_K]$ is a vector of boundaries. It is possible to learn the
 1953 weights and boundaries simultaneously, using a perceptron-like algorithm (Crammer and
 1954 Singer, 2001).

1955 **Lexicon-based classification** Sentiment analysis is one of the only NLP tasks where
 1956 hand-crafted feature weights are still widely employed. In **lexicon-based classification** (Taboada
 1957 et al., 2011), the user creates a list of words for each label, and then classifies each docu-
 1958 ment based on how many of the words from each list are present. In our linear classifica-
 1959 tion framework, this is equivalent to choosing the following weights:

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

1960 where \mathcal{L}_y is the lexicon for label y . Compared to the machine learning classifiers discussed
 1961 in the previous chapters, lexicon-based classification may seem primitive. However, su-
 1962 pervised machine learning relies on large annotated datasets, which are time-consuming
 1963 and expensive to produce. If the goal is to distinguish two or more categories in a new
 1964 domain, it may be simpler to start by writing down a list of words for each category.

1965 An early lexicon was the *General Inquirer* (Stone, 1966). Today, popular sentiment lex-
 1966 cons include SENTIWORDNET (Esuli and Sebastiani, 2006) and an evolving set of lexicons
 1967 from Liu (2015). For emotions and more fine-grained analysis, *Linguistic Inquiry and Word*
 1968 *Count* (LIWC) provides a set of lexicons (Tausczik and Pennebaker, 2010). The MPQA lex-
 1969 icon indicates the polarity (positive or negative) of 8221 terms, as well as whether they are
 1970 strongly or weakly subjective (Wiebe et al., 2005). A comprehensive comparison of senti-
 1971 ment lexicons is offered by Ribeiro et al. (2016). Given an initial **seed lexicon**, it is possible
 1972 to automatically expand the lexicon by looking for words that frequently co-occur with
 1973 words in the seed set (Hatzivassiloglou and McKeown, 1997; Qiu et al., 2011).

1974 4.2 Word sense disambiguation

1975 Consider the the following headlines:

- 1976 (4.8) Iraqi head seeks arms
- 1977 (4.9) Prostitutes appeal to Pope
- 1978 (4.10) Drunk gets nine years in violin case²

1979 These headlines are ambiguous because they contain words that have multiple mean-
 1980 ings, or **senses**. Word sense disambiguation is the problem of identifying the intended
 1981 sense of each word token in a document. Word sense disambiguation is part of a larger
 1982 field of research called **lexical semantics**, which is concerned with meanings of the words.

1983 At a basic level, the problem of word sense disambiguation is to identify the correct
 1984 sense for each word token in a document. Part-of-speech ambiguity (e.g., noun versus
 1985 verb) is usually considered to be a different problem, to be solved at an earlier stage.
 1986 From a linguistic perspective, senses are not properties of words, but of **lemmas**, which
 1987 are canonical forms that stand in for a set of inflected words. For example, *arm*/N is a
 1988 lemma that includes the inflected form *arms*/N — the /N indicates that it we are refer-
 1989 ring to the noun, and not its **homonym** *arm*/V, which is another lemma that includes
 1990 the inflected verbs (*arm*/V, *arms*/V, *armed*/V, *arming*/V). Therefore, word sense disam-
 1991 biguation requires first identifying the correct part-of-speech and lemma for each token,

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

1992 and then choosing the correct sense from the inventory associated with the corresponding
 1993 lemma.³ (Part-of-speech tagging is discussed in § 8.1.)

1994 **4.2.1 How many word senses?**

1995 Words sometimes have many more than two senses, as exemplified by the word *serve*:

- 1996 • [FUNCTION]: *The tree stump served as a table*
- 1997 • [CONTRIBUTE TO]: *His evasive replies only served to heighten suspicion*
- 1998 • [PROVIDE]: *We serve only the rawest fish*
- 1999 • [ENLIST]: *She served in an elite combat unit*
- 2000 • [JAIL]: *He served six years for a crime he didn't commit*
- 2001 • [LEGAL]: *They were served with subpoenas⁴*

2002 These sense distinctions are annotated in WORDNET (<http://wordnet.princeton.edu>).
 2003 a lexical semantic database for English. WORDNET consists of roughly 100,000
 2004 **synsets**, which are groups of lemmas (or phrases) that are synonymous. An example
 2005 synset is {*chump*¹, *fool*², *sucker*¹, *mark*⁹}, where the superscripts index the sense of each
 2006 lemma that is included in the synset: for example, there are at least eight other senses of
 2007 *mark* that have different meanings, and are not part of this synset. A lemma is **polysemous**
 2008 if it participates in multiple synsets.

2009 WORDNET defines the scope of the word sense disambiguation problem, and, more
 2010 generally, formalizes lexical semantic knowledge of English. (WordNets have been cre-
 2011 ated for a few dozen other languages, at varying levels of detail.) Some have argued
 2012 that WordNet's sense granularity is too fine (Ide and Wilks, 2006); more fundamentally,
 2013 the premise that word senses can be differentiated in a task-neutral way has been criti-
 2014 cized as linguistically naïve (Kilgarriff, 1997). One way of testing this question is to ask
 2015 whether people tend to agree on the appropriate sense for example sentences: accord-
 2016 ing to Mihalcea et al. (2004), people agree on roughly 70% of examples using WordNet
 2017 senses; far better than chance, but less than agreement on other tasks, such as sentiment
 2018 annotation (Wilson et al., 2005).

2019 ***Other lexical semantic relations** Besides **synonymy**, WordNet also describes many
 2020 other lexical semantic relationships, including:

- 2021 • **antonymy**: *x* means the opposite of *y*, e.g. FRIEND-ENEMY;

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

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

- 2022 • **hyponymy:** x is a special case of y , e.g. RED-COLOR; the inverse relationship is
2023 **hypernymy**;
2024 • **meronymy:** x is a part of y , e.g., WHEEL-BICYCLE; the inverse relationship is **holonymy**.

2025 Classification of these relations can be performed by searching for characteristic pat-
2026 terns between pairs of words, e.g., X , *such as* Y , which signals hyponymy (Hearst, 1992),
2027 or X *but* Y , which signals antonymy (Hatzivassiloglou and McKeown, 1997). Another ap-
2028 proach is to analyze each term's **distributional statistics** (the frequency of its neighboring
2029 words). Such approaches are described in detail in chapter 14.

2030 **4.2.2 Word sense disambiguation as classification**

2031 How can we tell living *plants* from manufacturing *plants*? The context is often critical:

- 2032 (4.11) Town officials are hoping to attract new manufacturing plants through weakened
2033 environmental regulations.
2034 (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,

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

2035 As in document classification, many of these features are irrelevant, but a few are very
2036 strong predictors. In this example, the context word *endangered* is a strong signal that
2037 the intended sense is biology rather than manufacturing. We would therefore expect a
2038 learning algorithm to assign high weight to (*endangered*, BIOLOGY), and low weight to
2039 (*endangered*, MANUFACTURING).⁵

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

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

2040 These are called **collocation features**, and they give more information about the specific
2041 role played by each context word. This idea can be taken further by incorporating addi-
2042 tional syntactic information about the grammatical role played by each context feature,
2043 such as the **dependency path** (see chapter 11).

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

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

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

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

As always, supervised classification is only possible if enough labeled examples can be accumulated. This is difficult in word sense disambiguation, because each polysemous lemma requires its own training set: having a good classifier for the senses of *serve* is no help towards disambiguating *plant*. For this reason, unsupervised and **semi-supervised** 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). Semi-supervised methods combine labeled and unlabeled data, and are discussed in more detail in chapter 5.

4.3 Design decisions for text classification

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

4.3.1 What is a word?

The bag-of-words representation presupposes that extracting a vector of word counts from text is unambiguous. But text documents are generally represented as a 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.

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

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

2075 Tokenization

2076 The first subtask for constructing a bag-of-words vector is **tokenization**: converting the
 2077 text from a sequence of characters to a sequence of **word!tokens**. A simple approach is
 2078 to define a subset of characters as whitespace, and then split the text on these tokens.
 2079 However, whitespace-based tokenization is not ideal: we may want to split conjunctions
 2080 like *isn't* and hyphenated phrases like *prize-winning* and *half-asleep*, and we likely want
 2081 to separate words from commas and periods that immediately follow them. At the same
 2082 time, it would be better not to split abbreviations like *U.S.* and *Ph.D.* In languages with
 2083 Roman scripts, tokenization is typically performed using regular expressions, with mod-
 2084 ules designed to handle each of these cases. For example, the NLTK package includes a
 2085 number of tokenizers (Loper and Bird, 2002); the outputs of four of the better-known tok-
 2086 enizers are shown in Figure 4.1. Social media researchers have found that emoticons and
 2087 other forms of orthographic variation pose new challenges for tokenization, leading to the
 2088 development of special purpose tokenizers to handle these phenomena (O'Connor et al.,
 2089 2010).

2090 Tokenization is a language-specific problem, and each language poses unique chal-
 2091 lenges. For example, Chinese does not include spaces between words, nor any other
 2092 consistent orthographic markers of word boundaries. A “greedy” approach is to scan the
 2093 input for character substrings that are in a predefined lexicon. However, Xue et al. (2003)
 2094 notes that this can be ambiguous, since many character sequences could be segmented in
 2095 multiple ways. Instead, he trains a classifier to determine whether each Chinese character,
 2096 or **hanzi**, is a word boundary. More advanced sequence labeling methods for word seg-
 2097 mentation are discussed in § 8.4. Similar problems can occur in languages with alphabetic
 2098 scripts, such as German, which does not include whitespace in compound nouns, yield-
 2099 ing examples such as *Freundschaftsbezeugungen* (demonstration of friendship) and *Dilett-*
 2100 *tantenaufdringlichkeiten* (the importunities of dilettantes). As Twain (1997) argues, “*These*
 2101 *things are not words, they are alphabetic processions.*” Social media raises similar problems
 2102 for English and other languages, with hashtags such as *#TrueLoveInFourWords* requiring
 2103 decomposition for analysis (Brun and Roux, 2014).

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

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

2104 **Text normalization**

2105 After splitting the text into tokens, the next question is which tokens are really distinct.
 2106 Is it necessary to distinguish *great*, *Great*, and *GREAT*? Sentence-initial capitalization may
 2107 be irrelevant to the classification task. Going further, the complete elimination of case
 2108 distinctions will result in a smaller vocabulary, and thus smaller feature vectors. However,
 2109 case distinctions might be relevant in some situations: for example, *apple* is a delicious
 2110 pie filling, while *Apple* is a company that specializes in proprietary dongles and power
 2111 adapters.

2112 For Roman script, case conversion can be performed using unicode string libraries.
 2113 Many scripts do not have case distinctions (e.g., the Devanagari script used for South
 2114 Asian languages, the Thai alphabet, and Japanese kana), and case conversion for all scripts
 2115 may not be available in every programming environment. (Unicode support is an im-
 2116 portant distinction between Python’s versions 2 and 3, and is a good reason for mi-
 2117 grating to Python 3 if you have not already done so. Compare the output of the code
 2118 "`\à l\hôtel`".upper()) in the two language versions.)⁶

2119 Case conversion is a type of **text normalization**, which refers to string transfor-
 2120 mations that remove distinctions that are irrelevant to downstream applications (Sproat et al.,
 2121 2001). Other forms of normalization include the standardization of numbers (e.g., 1,000 to
 2122 1000) and dates (e.g., August 11, 2015 to 2015/11/08). Depending on the application, it may
 2123 even be worthwhile to convert all numbers and dates to special tokens, !NUM and !DATE.
 2124 In social media, there are additional orthographic phenomena that may be normalized,
 2125 such as expressive lengthening, e.g., *coooooool* (Aw et al., 2006; Yang and Eisenstein, 2013).
 2126 Similarly, historical texts feature spelling variations that may need to be normalized to a
 2127 contemporary standard form (Baron and Rayson, 2008).

2128 A more extreme form of normalization is to eliminate **inflectional affixes**, such as the
 2129 -*ed* and -*s* suffixes in English. On this view, *bike*, *bikes*, *biking*, and *biked* all refer to the
 2130 same underlying concept, so they should be grouped into a single feature. A **stemmer** is
 2131 a program for eliminating affixes, usually by applying a series of regular expression sub-
 2132 stitutions. Character-based stemming algorithms are necessarily approximate, as shown

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

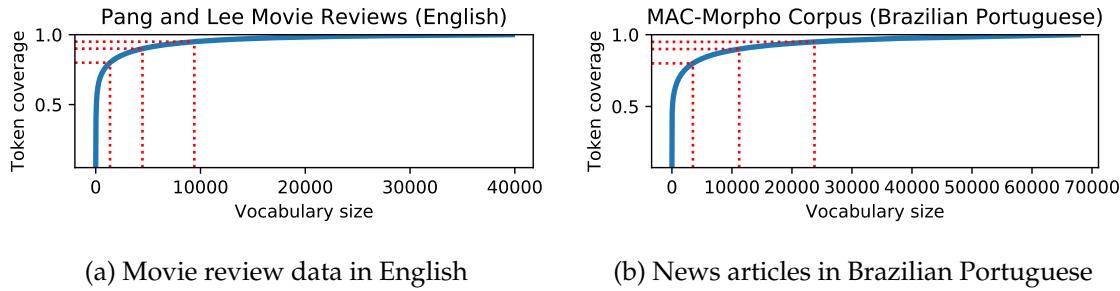


Figure 4.3: Tradeoff between token coverage (y-axis) and vocabulary size, on the NLTK movie review dataset, after sorting the vocabulary by decreasing frequency. The red dashed lines indicate 80%, 90%, and 95% coverage.

2133 in Figure 4.2: the Lancaster stemmer incorrectly identifies *-ers* as an inflectional suffix of
 2134 *sisters* (by analogy to *fix/fixers*), and both stemmers incorrectly identify *-s* as a suffix of *this*
 2135 and *Williams*. Fortunately, even inaccurate stemming can improve bag-of-words classifi-
 2136 cation models, by merging related strings and thereby reducing the vocabulary size.

2137 Accurately handling irregular orthography requires word-specific rules. **Lemmatizers**
 2138 are systems that identify the underlying lemma of a given wordform. They must avoid the
 2139 over-generalization errors of the stemmers in Figure 4.2, and also handle more complex
 2140 transformations, such as *geese*→*goose*. The output of the WordNet lemmatizer is shown in
 2141 the final line of Figure 4.2. Both stemming and lemmatization are language-specific: an
 2142 English stemmer or lemmatizer is of little use on a text written in another language. The
 2143 discipline of **morphology** relates to the study of word-internal structure, and is described
 2144 in more detail in § 9.1.2.

2145 The value of normalization depends on the data and the task. Normalization re-
 2146 duces the size of the feature space, which can help in generalization. However, there
 2147 is always the risk of merging away linguistically meaningful distinctions. In supervised
 2148 machine learning, regularization and smoothing can play a similar role to normalization
 2149 — preventing the learner from overfitting to rare features — while avoiding the language-
 2150 specific engineering required for accurate normalization. In unsupervised scenarios, such
 2151 as content-based information retrieval (Manning et al., 2008) and topic modeling (Blei
 2152 et al., 2003), normalization is more critical.

2153 4.3.2 How many words?

2154 Limiting the size of the feature vector reduces the memory footprint of the resulting mod-
 2155 els, and increases the speed of prediction. Normalization can help to play this role, but
 2156 a more direct approach is simply to limit the vocabulary to the N most frequent words
 2157 in the dataset. For example, in the MOVIE-REVIEWS dataset provided with NLTK (origi-

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

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

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

4.3.3 Count or binary?

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

2192 **4.4 Evaluating classifiers**

2193 In any supervised machine learning application, it is critical to reserve a held-out test set.
 2194 This data should be used for only one purpose: to evaluate the overall accuracy of a single
 2195 classifier. Using this data more than once would cause the estimated accuracy to be overly
 2196 optimistic, because the classifier would be customized to this data, and would not perform
 2197 as well as on unseen data in the future. It is usually necessary to set hyperparameters or
 2198 perform feature selection, so you may need to construct a **tuning** or **development set** for
 2199 this purpose, as discussed in § 2.2.5.

2200 There are a number of ways to evaluate classifier performance. The simplest is **accuracy**:
 2201 the number of correct predictions, divided by the total number of instances,

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

2202 Exams are usually graded by accuracy. Why are other metrics necessary? The main
 2203 reason is **class imbalance**. Suppose you are building a classifier to detect whether an
 2204 electronic health record (EHR) describes symptoms of a rare disease, which appears in
 2205 only 1% of all documents in the dataset. A classifier that reports $\hat{y} = \text{NEGATIVE}$ for
 2206 all documents would achieve 99% accuracy, but would be practically useless. We need
 2207 metrics that are capable of detecting the classifier's ability to discriminate between classes,
 2208 even when the distribution is skewed.

2209 One solution is to build a **balanced test set**, in which each possible label is equally rep-
 2210 resented. But in the EHR example, this would mean throwing away 98% of the original
 2211 dataset! Furthermore, the detection threshold itself might be a design consideration: in
 2212 health-related applications, we might prefer a very sensitive classifier, which returned a
 2213 positive prediction if there is even a small chance that $y^{(i)} = \text{POSITIVE}$. In other applica-
 2214 tions, a positive result might trigger a costly action, so we would prefer a classifier that
 2215 only makes positive predictions when absolutely certain. We need additional metrics to
 2216 capture these characteristics.

2217 **4.4.1 Precision, recall, and F-MEASURE**

2218 For any label (e.g., positive for presence of symptoms of a disease), there are two possible
 2219 errors:

- 2220 • **False positive**: the system incorrectly predicts the label.
 2221 • **False negative**: the system incorrectly fails to predict the label.

2222 Similarly, for any label, there are two ways to be correct:

- 2223 • **True positive:** the system correctly predicts the label.
 2224 • **True negative:** the system correctly predicts that the label does not apply to this
 2225 instance.

Classifiers that make a lot of false positives have low **precision**: they predict the label even when it isn't there. Classifiers that make a lot of false negatives have low **recall**: they fail to predict the label, even when it is there. These metrics distinguish these two sources of error, and are defined formally as:

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

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

2226 Recall and precision are both conditional likelihoods of a correct prediction, which is why
 2227 their numerators are the same. Recall is conditioned on k being the correct label, $y^{(i)} = k$,
 2228 so the denominator sums over true positive and false negatives. Precision is conditioned
 2229 on k being the prediction, so the denominator sums over true positives and false positives.
 2230 Note that true negatives are not considered in either statistic. The classifier that labels
 2231 every document as "negative" would achieve zero recall; precision would be $\frac{0}{0}$.

2232 Recall and precision are complementary. A high-recall classifier is preferred when
 2233 false positives are cheaper than false negatives: for example, in a preliminary screening
 2234 for symptoms of a disease, the cost of a false positive might be an additional test, while a
 2235 false negative would result in the disease going untreated. Conversely, a high-precision
 2236 classifier is preferred when false positives are more expensive: for example, in spam de-
 2237 tection, a false negative is a relatively minor inconvenience, while a false positive might
 2238 mean that an important message goes unread.

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

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

2239 where r is recall and p is precision.⁷

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

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

across each class. **Macro F-MEASURE** is the average *F*-MEASURE across several classes,

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

2240 In multi-class problems with unbalanced class distributions, the macro *F*-MEASURE is a
 2241 balanced measure of how well the classifier recognizes each class. In **micro F-MEASURE**,
 2242 we compute true positives, false positives, and false negatives for each class, and then add
 2243 them up to compute a single recall, precision, and *F*-MEASURE. This metric is balanced
 2244 across instances rather than classes, so it weights each class in proportion to its frequency
 2245 — unlike macro *F*-MEASURE, which weights each class equally.

2246 4.4.2 Threshold-free metrics

2247 In binary classification problems, it is possible to trade off between recall and precision by
 2248 adding a constant “threshold” to the output of the scoring function. This makes it possible
 2249 to trace out a curve, where each point indicates the performance at a single threshold. In
 2250 the **receiver operating characteristic (ROC)** curve,⁸ the *x*-axis indicates the **false positive**
 2251 **rate**, $\frac{FP}{FP+TN}$, and the *y*-axis indicates the recall, or **true positive rate**. A perfect classifier
 2252 attains perfect recall without any false positives, tracing a “curve” from the origin (0,0) to
 2253 the upper left corner (0,1), and then to (1,1). In expectation, a non-discriminative classifier
 2254 traces a diagonal line from the origin (0,0) to the upper right corner (1,1). Real classifiers
 2255 tend to fall between these two extremes. Examples are shown in Figure 4.4.

2256 The ROC curve can be summarized in a single number by taking its integral, the **area**
 2257 **under the curve (AUC)**. The AUC can be interpreted as the probability that a randomly-
 2258 selected positive example will be assigned a higher score by the classifier than a randomly-
 2259 selected negative example. A perfect classifier has AUC = 1 (all positive examples score
 2260 higher than all negative examples); a non-discriminative classifier has AUC = 0.5 (given
 2261 a randomly selected positive and negative example, either could score higher with equal
 2262 probability); a perfectly wrong classifier would have AUC = 0 (all negative examples score
 2263 higher than all positive examples). One advantage of AUC in comparison to *F*-MEASURE
 2264 is that the baseline rate of 0.5 does not depend on the label distribution.

2265 4.4.3 Classifier comparison and statistical significance

2266 Natural language processing research and engineering often involves comparing different
 2267 classification techniques. In some cases, the comparison is between algorithms, such as
 2268 logistic regression versus averaged perceptron, or L_2 regularization versus L_1 . In other

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

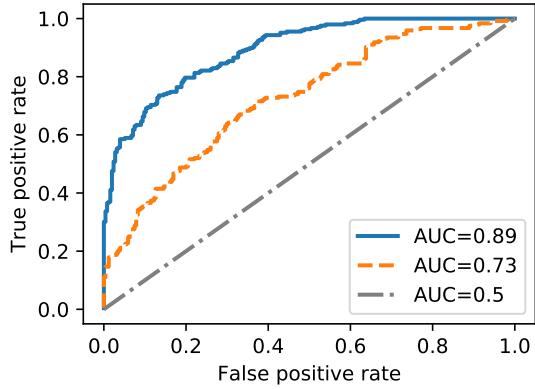


Figure 4.4: ROC curves for three classifiers of varying discriminative power, measured by AUC (area under the curve)

cases, the comparison is between feature sets, such as the bag-of-words versus positional bag-of-words (see § 4.2.2). **Ablation testing** involves systematically removing (ablating) various aspects of the classifier, such as feature groups, and testing the **null hypothesis** that the ablated classifier is as good as the full model.

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

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

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

2286 **The binomial test**

2287 The statistical significance of a difference in accuracy can be evaluated using classical tests,
 2288 such as the **binomial test**.¹¹ Suppose that classifiers c_1 and c_2 disagree on N instances in a
 2289 test set with binary labels, and that c_1 is correct on k of those instances. Under the null hy-
 2290 pothesis that the classifiers are equally accurate, we would expect k/N to be roughly equal
 2291 to $1/2$, and as N increases, k/N should be increasingly close to this expected value. These
 2292 properties are captured by the **binomial distribution**, which is a probability over counts
 2293 of binary random variables. We write $k \sim \text{Binom}(\theta, N)$ to indicate that k is drawn from
 2294 a binomial distribution, with parameter N indicating the number of random “draws”,
 2295 and θ indicating the probability of “success” on each draw. Each draw is an example on
 2296 which the two classifiers disagree, and a “success” is a case in which c_1 is right and c_2 is
 2297 wrong. (The label space is assumed to be binary, so if the classifiers disagree, exactly one
 2298 of them is correct. The test can be generalized to multi-class classification by focusing on
 2299 the examples in which exactly one classifier is correct.)

2300 The **probability mass function** (PMF) of the binomial distribution is,

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

2301 with θ^k representing the probability of the k successes, $(1 - \theta)^{N-k}$ representing the prob-
 2302 ability of the $N - k$ unsuccessful draws. The expression $\binom{N}{k} = \frac{N!}{k!(N-k)!}$ is a binomial
 2303 coefficient, representing the number of possible orderings of events; this ensures that the
 2304 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 $i \leq k$,

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

2305 The one-tailed p-value applies only to the asymmetric null hypothesis that c_1 is at least
 2306 as accurate as c_2 . To test the **two-tailed** null hypothesis that c_1 and c_2 are equally accu-
 2307 rate, we would take the sum of one-tailed p-values, where the second term is computed

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

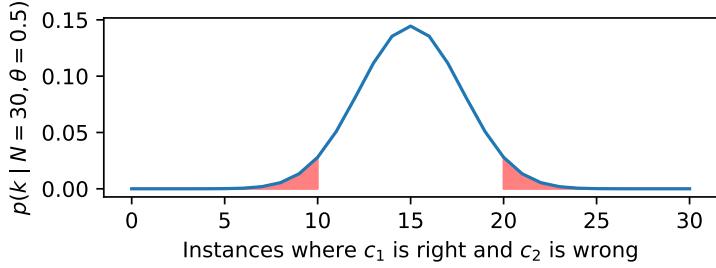


Figure 4.5: Probability mass function for the binomial distribution. The pink highlighted areas represent the cumulative probability for a significance test on an observation of $k = 10$ and $N = 30$.

from the right tail of Figure 4.5. The binomial distribution is symmetric, so this can be computed by simply doubling the one-tailed p-value.

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

*Randomized testing

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

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

Algorithm 7 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  $\{d^{(t)}\}_{t=1}^M$ 
```

2333 Algorithm 7, and then setting the top and bottom of the 95% confidence interval to the
 2334 values at the 2.5% and 97.5% percentiles of the sorted outputs. Alternatively, you can fit
 2335 a normal distribution to the set of differences across bootstrap samples, and compute a
 2336 Gaussian confidence interval from the mean and variance.

2337 As the number of bootstrap samples goes to infinity, $M \rightarrow \infty$, the bootstrap estimate
 2338 is increasingly accurate. A typical choice for M is 10^4 or 10^5 ; larger numbers of samples
 2339 are necessary for smaller p -values. One way to validate your choice of M is to run the test
 2340 multiple times, and ensure that the p -values are similar; if not, increase M by an order of
 2341 magnitude. This is a heuristic measure of the **variance** of the test, which can decreases
 2342 with the square root \sqrt{M} (Robert and Casella, 2013).

2343 **4.4.4 *Multiple comparisons**

2344 Sometimes it is necessary to perform multiple hypothesis tests, such as when comparing
 2345 the performance of several classifiers on multiple datasets. Suppose you have five
 2346 datasets, and you compare four versions of your classifier against a baseline system, for a
 2347 total of 20 comparisons. Even if none of your classifiers is better than the baseline, there
 2348 will be some chance variation in the results, and in expectation you will get one statistically
 2349 significant improvement at $p = 0.05 = \frac{1}{20}$. It is therefore necessary to adjust the
 2350 p -values when reporting the results of multiple comparisons.

2351 One approach is to require a threshold of $\frac{\alpha}{m}$ to report a p value of $p < \alpha$ when performing m tests. This is known as the **Bonferroni correction**, and it limits the overall probability of incorrectly rejecting the null hypothesis at α . Another approach is to bound the **false discovery rate** (FDR), which is the fraction of null hypothesis rejections that are incorrect. Benjamini and Hochberg (1995) propose a p -value correction that bounds the fraction of false discoveries at α : sort the p -values of each individual test in ascending order, and set the significance threshold equal to largest k such that $p_k \leq \frac{k}{m}\alpha$. If $k > 1$, the FDR adjustment is more permissive than the Bonferroni correction.

2359 **4.5 Building datasets**

2360 Sometimes, if you want to build a classifier, you must first build a dataset of your own.
 2361 This includes selecting a set of documents or instances to annotate, and then performing
 2362 the annotations. The scope of the dataset may be determined by the application: if you
 2363 want to build a system to classify electronic health records, then you must work with a
 2364 corpus of records of the type that your classifier will encounter when deployed. In other
 2365 cases, the goal is to build a system that will work across a broad range of documents. In
 2366 this case, it is best to have a *balanced* corpus, with contributions from many styles and
 2367 genres. For example, the Brown corpus draws from texts ranging from government doc-
 2368 uments to romance novels (Francis, 1964), and the Google Web Treebank includes an-
 2369 notations for five “domains” of web documents: question answers, emails, newsgroups,
 2370 reviews, and blogs (Petrov and McDonald, 2012).

2371 **4.5.1 Metadata as labels**

2372 Annotation is difficult and time-consuming, and most people would rather avoid it. It
 2373 is sometimes possible to exploit existing metadata to obtain labels for training a classi-
 2374 fier. For example, reviews are often accompanied by a numerical rating, which can be
 2375 converted into a classification label (see § 4.1). Similarly, the nationalities of social media
 2376 users can be estimated from their profiles (Dredze et al., 2013) or even the time zones of
 2377 their posts (Gouws et al., 2011). More ambitiously, we may try to classify the political af-
 2378 filiations of social media profiles based on their social network connections to politicians
 2379 and major political parties (Rao et al., 2010).

2380 The convenience of quickly constructing large labeled datasets without manual an-
 2381 notation is appealing. However this approach relies on the assumption that unlabeled
 2382 instances — for which metadata is unavailable — will be similar to labeled instances.
 2383 Consider the example of labeling the political affiliation of social media users based on
 2384 their network ties to politicians. If a classifier attains high accuracy on such a test set,
 2385 is it safe to assume that it accurately predicts the political affiliation of all social media
 2386 users? Probably not. Social media users who establish social network ties to politicians
 2387 may be more likely to mention politics in the text of their messages, as compared to the
 2388 average user, for whom no political metadata is available. If so, the accuracy on a test set
 2389 constructed from social network metadata would give an overly optimistic picture of the
 2390 method’s true performance on unlabeled data.

2391 **4.5.2 Labeling data**

2392 In many cases, there is no way to get ground truth labels other than manual annotation.
 2393 An annotation protocol should satisfy several criteria: the annotations should be *expressive*
 2394 enough to capture the phenomenon of interest; they should be *replicable*, meaning that

2395 another annotator or team of annotators would produce very similar annotations if given
2396 the same data; and they should be *scalable*, so that they can be produced relatively quickly.
2397 Hovy and Lavid (2010) propose a structured procedure for obtaining annotations that
2398 meet these criteria, which is summarized below.

- 2399 1. **Determine what the annotations are to include.** This is usually based on some
2400 theory of the underlying phenomenon: for example, if the goal is to produce annotations
2401 about the emotional state of a document’s author, one should start with a theoretical account
2402 of the types or dimensions of emotion (e.g., Mohammad and Turney, 2013). At this stage, the tradeoff
2403 between expressiveness and scalability should be considered: a full instantiation of the underlying theory might be too costly to
2404 annotate at scale, so reasonable approximations should be considered.
- 2406 2. Optionally, one may **design or select a software tool to support the annotation effort.** Existing general-purpose annotation tools include BRAT (Stenetorp et al.,
2407 2012) and MMAX2 (Müller and Strube, 2006).
- 2409 3. **Formalize the instructions for the annotation task.** To the extent that the instructions
2410 are not explicit, the resulting annotations will depend on the intuitions of the
2411 annotators. These intuitions may not be shared by other annotators, or by the users
2412 of the annotated data. Therefore explicit instructions are critical to ensuring the annotations
2413 are replicable and usable by other researchers.
- 2414 4. **Perform a pilot annotation** of a small subset of data, with multiple annotators for
2415 each instance. This will give a preliminary assessment of both the replicability and
2416 scalability of the current annotation instructions. Metrics for computing the rate of
2417 agreement are described below. Manual analysis of specific disagreements should
2418 help to clarify the instructions, and may lead to modifications of the annotation task
2419 itself. For example, if two labels are commonly conflated by annotators, it may be
2420 best to merge them.
- 2421 5. **Annotate the data.** After finalizing the annotation protocol and instructions, the
2422 main annotation effort can begin. Some, if not all, of the instances should receive
2423 multiple annotations, so that inter-annotator agreement can be computed. In some
2424 annotation projects, instances receive many annotations, which are then aggregated
2425 into a “consensus” label (e.g., Danescu-Niculescu-Mizil et al., 2013). However, if the
2426 annotations are time-consuming or require significant expertise, it may be preferable
2427 to maximize scalability by obtaining multiple annotations for only a small subset of
2428 examples.
- 2429 6. **Compute and report inter-annotator agreement, and release the data.** In some
2430 cases, the raw text data cannot be released, due to concerns related to copyright or

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

2438 Measuring inter-annotator agreement

2439 To measure the replicability of annotations, a standard practice is to compute the extent to
 2440 which annotators agree with each other. If the annotators frequently disagree, this casts
 2441 doubt on either their reliability or on the annotation system itself. For classification, one
 2442 can compute the frequency with which the annotators agree; for rating scales, one can
 2443 compute the average distance between ratings. These raw agreement statistics must then
 2444 be compared with the rate of agreement by chance — the expected level of agreement that
 2445 would be obtained between two annotators who ignored the data.

2446 **Cohen’s Kappa** is widely used for quantifying the agreement on discrete labeling
 2447 tasks (Cohen, 1960; Carletta, 1996),¹²

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

2448 The numerator is the difference between the observed agreement and the chance agree-
 2449 ment, and the denominator is the difference between perfect agreement and chance agree-
 2450 ment. Thus, $\kappa = 1$ when the annotators agree in every case, and $\kappa = 0$ when the annota-
 2451 tors agree only as often as would happen by chance. Various heuristic scales have been
 2452 proposed for determining when κ indicates “moderate”, “good”, or “substantial” agree-
 2453 ment; for reference, Lee and Narayanan (2005) report $\kappa \approx 0.45 - 0.47$ for annotations
 2454 of emotions in spoken dialogues, which they describe as “moderate agreement”; Stolcke
 2455 et al. (2000) report $\kappa = 0.8$ for annotations of **dialogue acts**, which are labels for the pur-
 2456 pose of each turn in a conversation.

2457 When there are two annotators, the expected chance agreement is computed as,

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

2458 where k is a sum over labels, and $\hat{\Pr}(Y = k)$ is the empirical probability of label k across
 2459 all annotations. The formula is derived from the expected number of agreements if the
 2460 annotations were randomly shuffled. Thus, in a binary labeling task, if one label is applied
 2461 to 90% of instances, chance agreement is $.9^2 + .1^2 = .82$.

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

2462 **Crowdsourcing**

2463 Crowdsourcing is often used to rapidly obtain annotations for classification problems.
 2464 For example, **Amazon Mechanical Turk** makes it possible to define “human intelligence
 2465 tasks (hits)”, such as labeling data. The researcher sets a price for each set of annotations
 2466 and a list of minimal qualifications for annotators, such as their native language and their
 2467 satisfaction rate on previous tasks. The use of relatively untrained “crowdworkers” con-
 2468 trasts with earlier annotation efforts, which relied on professional linguists (Marcus et al.,
 2469 1993). However, crowdsourcing has been found to produce reliable annotations for many
 2470 language-related tasks (Snow et al., 2008). Crowdsourcing is part of the broader field
 2471 of **human computation** (Law and Ahn, 2011). For a critical examination of ethical issues
 2472 related to crowdsourcing, see Fort et al. (2011).

2473 **Additional resources**

2474 Many of the preprocessing issues discussed in this chapter also arise in information re-
 2475 trieval. See Manning et al. (2008) for discussion of tokenization and related algorithms.

2476 **Exercises**

- 2477 1. As noted in § 4.3.3, words tend to appear in clumps, with subsequent occurrences
 2478 of a word being more probable. More concretely, if word j has probability $\phi_{y,j}$
 2479 of appearing in a document with label y , then the probability of two appearances
 2480 ($x_j^{(i)} = 2$) is greater than $\phi_{y,j}^2$.

2481 Suppose you are applying Naïve Bayes to a binary classification. Focus on a word j
 2482 which is more probable under label $y = 1$, so that,

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

2483 Now suppose that $x_j^{(i)} > 1$. All else equal, will the classifier overestimate or under-
 2484 estimate the posterior $\Pr(y = 1 \mid x)$?

- 2485 2. Prove that F-measure is never greater than the arithmetic mean of recall and preci-
 2486 sion, $\frac{r+p}{2}$. Your solution should also show that F-measure is equal to $\frac{r+p}{2}$ iff $r = p$.
 2487 3. Given a binary classification problem in which the probability of the “positive” label
 2488 is equal to α , what is the expected F-MEASURE of a random classifier which ignores
 2489 the data, and selects $\hat{y} = +1$ with probability $\frac{1}{2}$? (Assume that $p(\hat{y}) \perp p(y)$.) What is
 2490 the expected F-MEASURE of a classifier that selects $\hat{y} = +1$ with probability α (also
 2491 independent of $y^{(i)}$)? Depending on α , which random classifier will score better?

- 2492 4. Suppose that binary classifiers c_1 and c_2 disagree on $N = 30$ cases, and that c_1 is
 2493 correct in $k = 10$ of those cases.
- 2494 • Write a program that uses primitive functions such as `exp` and `factorial` to com-
 2495 pute the **two-tailed** p -value — you may use an implementation of the “choose”
 2496 function if one is available. Verify your code against the output of a library for
 2497 computing the binomial test or the binomial CDF, such as `SCIPY.STATS.BINOM`
 2498 in Python.
- 2499 • Then use a randomized test to try to obtain the same p -value. In each sample,
 2500 draw from a binomial distribution with $N = 30$ and $\theta = \frac{1}{2}$. Count the fraction
 2501 of samples in which $k \leq 10$. This is the one-tailed p -value; double this to
 2502 compute the two-tailed p -value.
- 2503 • Try this with varying numbers of bootstrap samples: $M \in \{100, 1000, 5000, 10000\}$.
 2504 For $M = 100$ and $M = 1000$, run the test 10 times, and plot the resulting p -
 2505 values.
- 2506 • Finally, perform the same tests for $N = 70$ and $k = 25$.
- 2507 5. SemCor 3.0 is a labeled dataset for word sense disambiguation. You can download
 2508 it,¹³ or access it in `NLTK.CORPORA.SEMCOR`.
- 2509 Choose a word that appears at least ten times in SemCor (*find*), and annotate its
 2510 WordNet senses across ten randomly-selected examples, without looking at the ground
 2511 truth. Use online WordNet to understand the definition of each of the senses.¹⁴ Have
 2512 a partner do the same annotations, and compute the raw rate of agreement, expected
 2513 chance rate of agreement, and Cohen’s kappa.
- 2514 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
 2515 randomly-selected 400 reviews as a test set.
- 2516 Download a sentiment lexicon, such as the one currently available from Bing Liu,
 2517 <https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html>. Tokenize
 2518 the data, and classify each document as positive iff it has more positive sentiment
 2519 words than negative sentiment words. Compute the accuracy and *F*-MEASURE on
 2520 detecting positive reviews on the test set, using this lexicon-based classifier.
- 2521 Then train a discriminative classifier (averaged perceptron or logistic regression) on
 2522 the training set, and compute its accuracy and *F*-MEASURE on the test set.
- 2523 Determine whether the differences are statistically significant, using two-tailed hy-
 2524 pothesis tests: Binomial for the difference in accuracy, and bootstrap for the differ-
 2525 ence in macro-*F*-MEASURE.

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

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

2527 The remaining problems will require you to build a classifier and test its properties. Pick
2528 a multi-class text classification dataset that is not already tokenized. One example is a
2529 dataset of New York Times headlines and topics (BoydStun, 2013).¹⁵ Divide your data
2530 into training (60%), development (20%), and test sets (20%), if no such division already
2531 exists. If your dataset is very large, you may want to focus on a few thousand instances at
2532 first.

2533 7. Compare various vocabulary sizes of 10^2 , 10^3 , 10^4 , 10^5 , using the most frequent words
2534 in each case (you may use any reasonable tokenizer). Train logistic regression clas-
2535 sifiers for each vocabulary size, and apply them to the development set. Plot the
2536 accuracy and Macro-*F*-MEASURE with the increasing vocabulary size. For each vo-
2537 cabulary size, tune the regularizer to maximize accuracy on a subset of data that is
2538 held out from the training set.

2539 8. Compare the following tokenization algorithms:

- 2540 • Whitespace, using a regular expression;
2541 • The Penn Treebank tokenizer from NLTK;
2542 • Splitting the input into non-overlapping five-character units, regardless of whites-
2543 pace or punctuation.

2544 Compute the token/type ratio for each tokenizer on the training data, and explain
2545 what you find. Train your classifier on each tokenized dataset, tuning the regularizer
2546 on a subset of data that is held out from the training data. Tokenize the development
2547 set, and report accuracy and Macro-*F*-MEASURE.

2548 9. Apply the Porter and Lancaster stemmers to the training set, using any reasonable
2549 tokenizer, and compute the token/type ratios. Train your classifier on the stemmed
2550 data, and compute the accuracy and Macro-*F*-MEASURE on stemmed development
2551 data, again using a held-out portion of the training data to tune the regularizer.

2552 10. Identify the best combination of vocabulary filtering, tokenization, and stemming
2553 from the previous three problems. Apply this preprocessing to the test set, and
2554 compute the test set accuracy and Macro-*F*-MEASURE. Compare against a baseline
2555 system that applies no vocabulary filtering, whitespace tokenization, and no stem-
2556 ming.

2557 Use the binomial test to determine whether your best-performing system is signifi-
2558 cantly more accurate than the baseline.

¹⁵Available as a CSV file at <http://www.amber-boydstun.com/supplementary-information-for-making-the-news.html>. Use the field TOPIC_2DIGIT for this problem.

2559 Use the bootstrap test with $M = 10^4$ to determine whether your best-performing
2560 system achieves significantly higher macro-*F*-MEASURE.

2561

Chapter 5

2562

Learning without supervision

2563 So far we've assumed the following setup:

- 2564 • a **training set** where you get observations x and labels y ;
2565 • a **test set** where you only get observations x .

2566 Without labeled data, is it possible to learn anything? This scenario is known as **unsu-**
2567 **pervised learning**, and we will see that indeed it is possible to learn about the underlying
2568 structure of unlabeled observations. This chapter will also explore some related scenarios:
2569 **semi-supervised learning**, in which only some instances are labeled, and **domain adap-**
2570 **tation**, in which the training data differs from the data on which the trained system will
2571 be deployed.

2572

5.1 Unsupervised learning

2573 To motivate unsupervised learning, consider the problem of word sense disambiguation
2574 (§ 4.2). The goal is to classify each instance of a word, such as *bank* into a sense,

- 2575 • bank#1: a financial institution
2576 • bank#2: the land bordering a river

2577 It is difficult to obtain sufficient training data for word sense disambiguation, because
2578 even a large corpus will contain only a few instances of all but the most common words.
2579 Is it possible to learn anything about these different senses without labeled data?

2580 Word sense disambiguation is usually performed using feature vectors constructed
2581 from the local context of the word to be disambiguated. For example, for the word

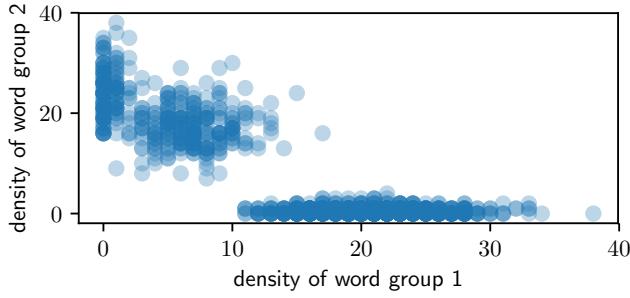


Figure 5.1: Counts of words from two different context groups

2582 *bank*, the immediate context might typically include words from one of the following two
 2583 groups:

- 2584 1. *financial, deposits, credit, lending, capital, markets, regulated, reserve, liquid, assets*
 2585 2. *land, water, geography, stream, river, flow, deposits, discharge, channel, ecology*

2586 Now consider a scatterplot, in which each point is a document containing the word *bank*.
 2587 The location of the document on the x -axis is the count of words in group 1, and the
 2588 location on the y -axis is the count for group 2. In such a plot, shown in Figure 5.1, two
 2589 “blobs” might emerge, and these blobs correspond to the different senses of *bank*.

2590 Here’s a related scenario, from a different problem. Suppose you download thousands
 2591 of news articles, and make a scatterplot, where each point corresponds to a document:
 2592 the x -axis is the frequency of the group of words (*hurricane, winds, storm*); the y -axis is the
 2593 frequency of the group (*election, voters, vote*). This time, three blobs might emerge: one
 2594 for documents that are largely about a hurricane, another for documents largely about a
 2595 election, and a third for documents about neither topic.

2596 These clumps represent the underlying structure of the data. But the two-dimensional
 2597 scatter plots are based on groupings of context words, and in real scenarios these word
 2598 lists are unknown. Unsupervised learning applies the same basic idea, but in a high-
 2599 dimensional space with one dimension for every context word. This space can’t be di-
 2600 rectly visualized, but the idea is the same: try to identify the underlying structure of the
 2601 observed data, such that there are a few clusters of points, each of which is internally
 2602 coherent. **Clustering** algorithms are capable of finding such structure automatically.

2603 5.1.1 **K-means** clustering

2604 Clustering algorithms assign each data point to a discrete cluster, $z_i \in 1, 2, \dots, K$. One of
 2605 the best known clustering algorithms is ***K-means***, an iterative algorithm that maintains

Algorithm 8 K -means clustering algorithm

```

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

```

2606 a cluster assignment for each instance, and a central (“mean”) location for each cluster.
 2607 K -means iterates between updates to the assignments and the centers:

2608 1. each instance is placed in the cluster with the closest center;

2609 2. each center is recomputed as the average over points in the cluster.

2610 This procedure is formalized in Algorithm 8. The term $\|\mathbf{x}^{(i)} - \boldsymbol{\nu}\|^2$ refers to the squared
 2611 Euclidean norm, $\sum_{j=1}^V (x_j^{(i)} - \nu_j)^2$. An important property of K -means is that the con-
 2612 verged solution depends on the initialization, and a better clustering can sometimes be
 2613 found simply by re-running the algorithm from a different random starting point.

2614 **Soft K -means** is a particularly relevant variant. Instead of directly assigning each
 2615 point to a specific cluster, soft K -means assigns to each point a *distribution* over clusters
 2616 $\mathbf{q}^{(i)}$, 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
 2617 the distance of $\mathbf{x}^{(i)}$ to the cluster center $\boldsymbol{\nu}_k$. In turn, the center of each cluster is computed
 2618 from a weighted average of the points in the cluster,

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

2619 We will now explore a probabilistic version of soft K -means clustering, based on **expectation-**
 2620 **maximization** (EM). Because EM clustering can be derived as an approximation to maximum-
 2621 likelihood estimation, it can be extended in a number of useful ways.

2622 5.1.2 Expectation-Maximization (EM)

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

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

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

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

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

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

2623 The parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$ can be obtained by maximizing the marginal likelihood in
 2624 Equation 5.5. Why is this the right thing to maximize? Without labels, discriminative
 2625 learning is impossible — there's nothing to discriminate. So maximum likelihood is all
 2626 we have.

2627 When the labels are observed, we can estimate the parameters of the Naïve Bayes
 2628 probability model separately for each label. But marginalizing over the labels couples
 2629 these parameters, making direct optimization of $\log p(\mathbf{x})$ intractable. We will approxi-
 2630 mate the log-likelihood by introducing an auxiliary variable $\mathbf{q}^{(i)}$, which is a distribution
 2631 over the label set $\mathcal{Z} = \{1, 2, \dots, K\}$. The optimization procedure will alternate between
 2632 updates to \mathbf{q} and updates to the parameters $(\boldsymbol{\phi}, \boldsymbol{\mu})$. Thus, $\mathbf{q}^{(i)}$ plays here as in soft K -
 2633 means.

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

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

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

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

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

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

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

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

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

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

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

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

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

2645 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.

2651 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,

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

2652 The term $\log p(\mathbf{x}^{(i)} | z; \boldsymbol{\phi})$ is the conditional log-likelihood for the multinomial, which
2653 expands to,

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

2654 where C is a constant with respect to $\boldsymbol{\phi}$ — see Equation 2.12 in § 2.2 for more discussion
2655 of this probability function.

Setting the derivative of J_ϕ equal to zero,

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

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

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

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

2656 where the counter $j \in \{1, 2, \dots, V\}$ indexes over base features, such as words.

2657 This update sets ϕ_z equal to the relative frequency estimate of the *expected counts* under
2658 the distribution q . As in supervised Naïve Bayes, we can smooth these counts by adding
2659 a constant α . The update for μ is similar: $\mu_z \propto \sum_{i=1}^N q^{(i)}(z) = E_q [\text{count}(z)]$, which is the
2660 expected frequency of cluster z . These probabilities can also be smoothed. In sum, the
2661 M-step is just like Naïve Bayes, but with expected counts rather than observed counts.

2662 The multinomial likelihood $p(\mathbf{x} | z)$ can be replaced with other probability distribu-
2663 tions: for example, for continuous observations, a Gaussian distribution can be used. In
2664 some cases, there is no closed-form update to the parameters of the likelihood. One ap-
2665 proach is to run gradient-based optimization at each M-step; another is to simply take a
2666 single step along the gradient step and then return to the E-step (Berg-Kirkpatrick et al.,
2667 2010).

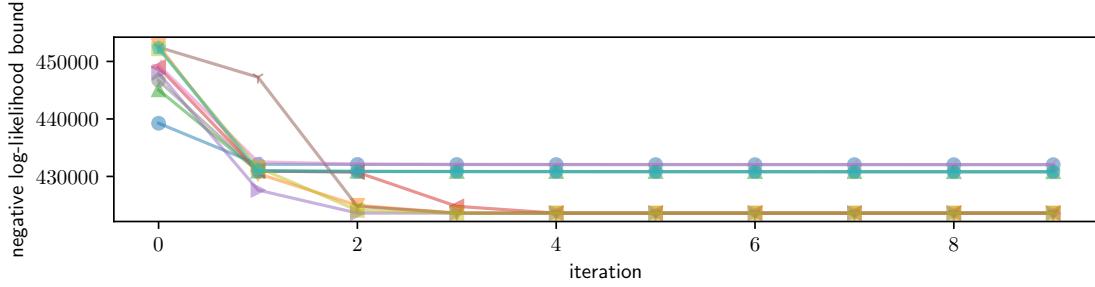


Figure 5.2: Sensitivity of expectation-maximization to initialization. Each line shows the progress of optimization from a different random initialization.

2668 5.1.3 EM as an optimization algorithm

2669 Algorithms that update a global objective by alternating between updates to subsets of the
 2670 parameters are called **coordinate ascent** algorithms. The objective J (the lower bound on
 2671 the marginal likelihood of the data) is separately convex in q and (μ, ϕ) , but it is not jointly
 2672 convex in all terms; this condition is known as **biconvexity**. Each step of the expectation-
 2673 maximization algorithm is guaranteed not to decrease the lower bound J , which means
 2674 that EM will converge towards a solution at which no nearby points yield further im-
 2675 provements. This solution is a **local optimum** — it is as good or better than any of its
 2676 immediate neighbors, but is *not* guaranteed to be optimal among all possible configura-
 2677 tions of (q, μ, ϕ) .

2678 The fact that there is no guarantee of global optimality means that initialization is
 2679 important: where you start can determine where you finish. To illustrate this point,
 2680 Figure 5.2 shows the objective function for EM with ten different random initializations:
 2681 while the objective function improves monotonically in each run, it converges to several
 2682 different values.¹ For the convex objectives that we encountered in chapter 2, it was not
 2683 necessary to worry about initialization, because gradient-based optimization guaranteed
 2684 to reach the global minimum. But in expectation-maximization — as in the deep neural
 2685 networks from chapter 3 — initialization matters.

2686 In **hard EM**, each $q^{(i)}$ distribution assigns probability of 1 to a single label $\hat{z}^{(i)}$, and zero
 2687 probability to all others (Neal and Hinton, 1998). This is similar in spirit to K -means clus-
 2688 tering, and can outperform standard EM in some cases (Spitkovsky et al., 2010). Another
 2689 variant of expectation-maximization incorporates stochastic gradient descent (SGD): after
 2690 performing a local E-step at each instance $x^{(i)}$, we immediately make a gradient update
 2691 to the parameters (μ, ϕ) . This algorithm has been called **incremental expectation maxi-
 2692 mization** (Neal and Hinton, 1998) and **online expectation maximization** (Sato and Ishii,

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

2693 2000; Cappé and Moulines, 2009), and is especially useful when there is no closed-form
 2694 optimum for the likelihood $p(\mathbf{x} | z)$, and in online settings where new data is constantly
 2695 streamed in (see Liang and Klein, 2009, for a comparison for online EM variants).

2696 **5.1.4 How many clusters?**

2697 So far, we have assumed that the number of clusters K is given. In some cases, this as-
 2698 sumption is valid. For example, a lexical semantic resource like WORDNET might define
 2699 the number of senses for a word. In other cases, the number of clusters could be a parame-
 2700 ter for the user to tune: some readers want a coarse-grained clustering of news stories into
 2701 three or four clusters, while others want a fine-grained clustering into twenty or more. But
 2702 many times there is little extrinsic guidance for how to choose K .

2703 One solution is to choose the number of clusters to maximize a metric of clustering
 2704 quality. The other parameters μ and ϕ are chosen to maximize the log-likelihood bound
 2705 J , so this might seem a potential candidate for tuning K . However, J will never decrease
 2706 with K : if it is possible to obtain a bound of J_K with K clusters, then it is always possible
 2707 to do at least as well with $K + 1$ clusters, by simply ignoring the additional cluster and
 2708 setting its probability to zero in q and μ . It is therefore necessary to introduce a penalty
 2709 for model complexity, so that fewer clusters are preferred. For example, the Akaike Infor-
 2710 mation Crition (AIC; Akaike, 1974) is the linear combination of the number of parameters
 2711 and the log-likelihood,

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

2712 where M is the number of parameters. In an expectation-maximization clustering algo-
 2713 rithm, $M = K \times V + K$. Since the number of parameters increases with the number of
 2714 clusters K , the AIC may prefer more parsimonious models, even if they do not fit the data
 2715 quite as well.

2716 Another choice is to maximize the **predictive likelihood** on heldout data. This data
 2717 is not used to estimate the model parameters ϕ and μ , and so it is not the case that the
 2718 likelihood on this data is guaranteed to increase with K . Figure 5.3 shows the negative
 2719 log-likelihood on training and heldout data, as well as the AIC.

2720 ***Bayesian nonparametrics** An alternative approach is to treat the number of clusters
 2721 as another latent variable. This requires statistical inference over a set of models with a
 2722 variable number of clusters. This is not possible within the framework of expecta-
 2723 tion maximization, but there are several alternative inference procedures which can be ap-
 2724 plied, including **Markov Chain Monte Carlo (MCMC)**, which is briefly discussed in
 2725 § 5.5 (for more details, see Chapter 25 of Murphy, 2012). Bayesian nonparametrics have
 2726 been applied to the problem of unsupervised word sense induction, learning not only the
 2727 word senses but also the number of senses per word (Reisinger and Mooney, 2010).

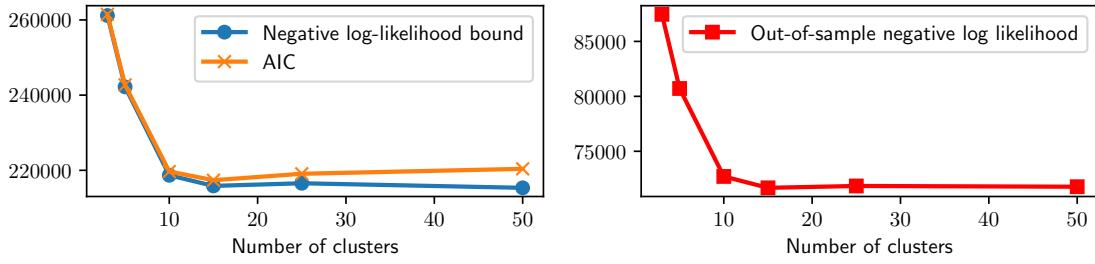


Figure 5.3: The negative log-likelihood and AIC for several runs of expectation-maximization, on synthetic data. Although the data was generated from a model with $K = 10$, the optimal number of clusters is $\hat{K} = 15$, according to AIC and the heldout log-likelihood. The training set log-likelihood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

2728 5.2 Applications of expectation-maximization

2729 EM is not really an “algorithm” like, say, quicksort. Rather, it is a framework for learning
2730 with missing data. The recipe for using EM on a problem of interest is:

- 2731 • Introduce latent variables z , such that it is easy to write the probability $P(\mathbf{x}, z)$. It
2732 should also be easy to estimate the associated parameters, given knowledge of z .
- 2733 • Derive the E-step updates for $q(z)$, which is typically factored as $q(z) = \prod_{i=1}^N q_{z^{(i)}}(z^{(i)})$,
2734 where i is an index over instances.
- 2735 • The M-step updates typically correspond to the soft version of a probabilistic super-
2736 vised learning algorithm, like Naïve Bayes.

2737 This section discusses a few of the many applications of this general framework.

2738 5.2.1 Word sense induction

2739 The chapter began by considering the problem of word sense disambiguation when the
2740 senses are not known in advance. Expectation-maximization can be applied to this prob-
2741 lem by treating each cluster as a word sense. Each instance represents the use of an
2742 ambiguous word, and $\mathbf{x}^{(i)}$ is a vector of counts for the other words that appear nearby:
2743 Schütze (1998) uses all words within a 50-word window. The probability $p(\mathbf{x}^{(i)} | z)$ can be
2744 set to the multinomial distribution, as in Naïve Bayes. The EM algorithm can be applied
2745 directly to this data, yielding clusters that (hopefully) correspond to the word senses.

Better performance can be obtained by first applying **singular value decomposition** (SVD) to the matrix of context-counts $\mathbf{C}_{ij} = \text{count}(i, j)$, where $\text{count}(i, j)$ is the count of word j in the context of instance i . **Truncated** singular value decomposition approximates

the matrix \mathbf{C} as a product of three matrices, $\mathbf{U}, \mathbf{S}, \mathbf{V}$, under the constraint that \mathbf{U} and \mathbf{V} are orthonormal, and \mathbf{S} is diagonal:

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

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

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

5.2.2 Semi-supervised learning

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

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

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

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

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

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

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

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

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

2765 The left sum is identical to the objective in Naïve Bayes; the right sum is the marginal log-
 2766 likelihood for expectation-maximization clustering, from Equation 5.5. We can construct a
 2767 lower bound on this log-likelihood by introducing distributions $q^{(j)}$ for all $j \in \{N_\ell + 1, \dots, N_\ell + N_u\}$.
 2768 The E-step updates these distributions; the M-step updates the parameters ϕ and μ , us-
 2769 ing the expected counts from the unlabeled data and the observed counts from the labeled
 2770 data.

2771 A critical issue in semi-supervised learning is how to balance the impact of the labeled
 2772 and unlabeled data on the classifier weights, especially when the unlabeled data is much
 2773 larger than the labeled dataset. The risk is that the unlabeled data will dominate, caus-
 2774 ing the parameters to drift towards a “natural clustering” of the instances — which may
 2775 not correspond to a good classifier for the labeled data. One solution is to heuristically
 2776 reweight the two components of Equation 5.26, tuning the weight of the two components
 2777 on a heldout development set (Nigam et al., 2000).

2778 **5.2.3 Multi-component modeling**

2779 As a final application, let’s return to fully supervised classification. A classic dataset for
 2780 text classification is 20 newsgroups, which contains posts to a set of online forums, called
 2781 newsgroups. One of the newsgroups is `comp.sys.mac.hardware`, which discusses Ap-
 2782 ple computing hardware. Suppose that within this newsgroup there are two kinds of
 2783 posts: reviews of new hardware, and question-answer posts about hardware problems.
 2784 The language in these *components* of the `mac.hardware` class might have little in com-
 2785 mon; if so, it would be better to model these components separately, rather than treating
 2786 their union as a single class. However, the component responsible for each instance is not
 2787 directly observed.

2788 Recall that Naïve Bayes is based on a generative process, which provides a stochastic
 2789 explanation for the observed data. In Naïve Bayes, each label is drawn from a categorical
 2790 distribution with parameter μ , and each vector of word counts is drawn from a multi-
 2791 nomial distribution with parameter ϕ_y . For multi-component modeling, we envision a
 2792 slightly different generative process, incorporating both the observed label $y^{(i)}$ and the
 2793 latent component $z^{(i)}$. This generative process is shown in Algorithm 9. A new parameter
 2794 $\beta_{y^{(i)}}$ defines the distribution of components, conditioned on the label $y^{(i)}$. The component,
 2795 and not the class label, then parametrizes the distribution over words.

-
- (5.1) ☺ Villeneuve a bel et bien **réussi** son pari de changer de perspectives tout en assurant une cohérence à la franchise.²
- (5.2) ☺ Il est également trop **long** et bancal dans sa narration, tiède dans ses intentions, et tirailé entre deux personnages et directions qui ne parviennent pas à coexister en harmonie.³
- (5.3) Denis Villeneuve a **réussi** une suite **parfaitemment** maîtrisée⁴
- (5.4) **Long, bavard**, hyper design, à peine agité (le comble de l'action : une bagarre dans la flotte), métaphysique et, surtout, ennuyeux jusqu'à la catalepsie.⁵
- (5.5) Une suite d'une écrasante puissance, mêlant **parfaitemment** le contemplatif au narratif.⁶
- (5.6) Le film impitoyablement **bavard** finit quand même par se taire quand se lève l'espèce de bouquet final où semble se déchaîner, comme en libre parcours de poulets décapiés, l'armée des graphistes numériques griffant nerveusement la palette graphique entre agonie et orgasme.⁷

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

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

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

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

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

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

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

²⁷⁹⁶ **5.3 Semi-supervised learning**

²⁷⁹⁷ In semi-supervised learning, the learner makes use of both labeled and unlabeled data.
²⁷⁹⁸ To see how this could help, suppose you want to do sentiment analysis in French. In Ta-

2799 ble 5.1, there are two labeled examples, one positive and one negative. From this data, a
 2800 learner could conclude that *réussi* is positive and *long* is negative. This isn't much! How-
 2801 ever, we can propagate this information to the unlabeled data, and potentially learn more.

- 2802 • If we are confident that *réussi* is positive, then we might guess that (5.3) is also posi-
 2803 tive.
- 2804 • That suggests that *parfaitement* is also positive.
- 2805 • We can then propagate this information to (5.5), and learn from this words in this
 2806 example.
- 2807 • Similarly, we can propagate from the labeled data to (5.4), which we guess to be
 2808 negative because it shares the word *long*. This suggests that *bavard* is also negative,
 2809 which we propagate to (5.6).

2810 Instances (5.3) and (5.4) were "similar" to the labeled examples for positivity and negativ-
 2811 ity, respectively. By using these instances to expand the models for each class, it became
 2812 possible to correctly label instances (5.5) and (5.6), which didn't share any important fea-
 2813 tures with the original labeled data. This requires a key assumption: that similar instances
 2814 will have similar labels.

2815 In § 5.2.2, we discussed how expectation-maximization can be applied to semi-supervised
 2816 learning. Using the labeled data, the initial parameters ϕ would assign a high weight for
 2817 *réussi* in the positive class, and a high weight for *long* in the negative class. These weights
 2818 helped to shape the distributions q for instances (5.3) and (5.4) in the E-step. In the next
 2819 iteration of the M-step, the parameters ϕ are updated with counts from these instances,
 2820 making it possible to correctly label the instances (5.5) and (5.6).

2821 However, expectation-maximization has an important disadvantage: it requires using
 2822 a generative classification model, which restricts the features that can be used for clas-
 2823 sification. In this section, we explore non-probabilistic approaches, which impose fewer
 2824 restrictions on the classification model.

2825 5.3.1 Multi-view learning

2826 EM semi-supervised learning can be viewed as **self-training**: the labeled data guides the
 2827 initial estimates of the classification parameters; these parameters are used to compute
 2828 a label distribution over the unlabeled instances, $q^{(i)}$; the label distributions are used to
 2829 update the parameters. The risk is that self-training drifts away from the original labeled
 2830 data. This problem can be ameliorated by **multi-view learning**. Here we take the as-
 2831 sumption that the features can be decomposed into multiple "views", each of which is
 2832 conditionally independent, given the label. For example, consider the problem of classi-
 2833 fying a name as a person or location: one view is the name itself; another is the context in
 2834 which it appears. This situation is illustrated in Table 5.2.

| | $\boldsymbol{x}^{(1)}$ | $\boldsymbol{x}^{(2)}$ | 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

2835 **Co-training** is an iterative multi-view learning algorithm, in which there are separate
 2836 classifiers for each view (Blum and Mitchell, 1998). At each iteration of the algorithm, each
 2837 classifier predicts labels for a subset of the unlabeled instances, using only the features
 2838 available in its view. These predictions are then used as ground truth to train the classifiers
 2839 associated with the other views. In the example shown in Table 5.2, the classifier on $\boldsymbol{x}^{(1)}$
 2840 might correctly label instance #5 as a person, because of the feature *Dr*; this instance would
 2841 then serve as training data for the classifier on $\boldsymbol{x}^{(2)}$, which would then be able to correctly
 2842 label instance #6, thanks to the feature *recommended*. If the views are truly independent,
 2843 this procedure is robust to drift. Furthermore, it imposes no restrictions on the classifiers
 2844 that can be used for each view.

2845 Word-sense disambiguation is particularly suited to multi-view learning, thanks to the
 2846 heuristic of “one sense per discourse”: if a polysemous word is used more than once in
 2847 a given text or conversation, all usages refer to the same sense (Gale et al., 1992). This
 2848 motivates a multi-view learning approach, in which one view corresponds to the local
 2849 context (the surrounding words), and another view corresponds to the global context at
 2850 the document level (Yarowsky, 1995). The local context view is first trained on a small
 2851 seed dataset. We then identify its most confident predictions on unlabeled instances. The
 2852 global context view is then used to extend these confident predictions to other instances
 2853 within the same documents. These new instances are added to the training data to the
 2854 local context classifier, which is retrained and then applied to the remaining unlabeled
 2855 data.

2856 5.3.2 Graph-based algorithms

2857 Another family of approaches to semi-supervised learning begins by constructing a graph,
 2858 in which pairs of instances are linked with symmetric weights $\omega_{i,j}$, e.g.,

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

2859 The goal is to use this weighted graph to propagate labels from a small set of labeled
 2860 instances to larger set of unlabeled instances.

2861 In **label propagation**, this is done through a series of matrix operations (Zhu et al.,
 2862 2003). Let \mathbf{Q} be a matrix of size $N \times K$, in which each row $\mathbf{q}^{(i)}$ describes the labeling
 2863 of instance i . When ground truth labels are available, then $\mathbf{q}^{(i)}$ is an indicator vector,
 2864 with $q_{y^{(i)}}^{(i)} = 1$ and $q_{y' \neq y^{(i)}}^{(i)} = 0$. Let us refer to the submatrix of rows containing labeled
 2865 instances as \mathbf{Q}_L , and the remaining rows as \mathbf{Q}_U . The rows of \mathbf{Q}_U are initialized to assign
 2866 equal probabilities to all labels, $q_{i,k} = \frac{1}{K}$.

2867 Now, let $T_{i,j}$ represent the “transition” probability of moving from node j to node i ,

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

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

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

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

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

2868 The expression $\tilde{\mathbf{Q}}_U \mathbf{1}$ indicates multiplication of $\tilde{\mathbf{Q}}_U$ by a column vector of ones, which is
 2869 equivalent to computing the sum of each row of $\tilde{\mathbf{Q}}_U$. The matrix $\text{Diag}(\mathbf{s})$ is a diagonal
 2870 matrix with the elements of \mathbf{s} on the diagonals. The product $\text{Diag}(\mathbf{s})^{-1} \tilde{\mathbf{Q}}_U$ has the effect
 2871 of normalizing the rows of $\tilde{\mathbf{Q}}_U$, so that each row of \mathbf{Q}_U is a probability distribution over
 2872 labels.

2873 5.4 Domain adaptation

2874 In many practical scenarios, the labeled data differs in some key respect from the data
 2875 to which the trained model is to be applied. A classic example is in consumer reviews:
 2876 we may have labeled reviews of movies (the source domain), but we want to predict the
 2877 reviews of appliances (the target domain). A similar issues arise with genre differences:
 2878 most linguistically-annotated data is news text, but application domains range from social
 2879 media to electronic health records. In general, there may be several source and target
 2880 domains, each with their own properties; however, for simplicity, this discussion will
 2881 focus mainly on the case of a single source and target domain.

2882 The simplest approach is “direct transfer”: train a classifier on the source domain, and
 2883 apply it directly to the target domain. The accuracy of this approach depends on the extent
 2884 to which features are shared across domains. In review text, words like *outstanding* and

2885 *disappointing* will apply across both movies and appliances; but others, like *terrifying*, may
 2886 have meanings that are domain-specific. As a result, direct transfer performs poorly: for
 2887 example, an out-of-domain classifier (trained on book reviews) suffers twice the error rate
 2888 of an in-domain classifier on reviews of kitchen appliances (Blitzer et al., 2007). **Domain**
 2889 **adaptation** algorithms attempt to do better than direct transfer by learning from data in
 2890 both domains. There are two main families of domain adaptation algorithms, depending
 2891 on whether any labeled data is available in the target domain.

2892 5.4.1 Supervised domain adaptation

2893 In supervised domain adaptation, there is a small amount of labeled data in the target
 2894 domain, and a large amount of data in the source domain. The simplest approach would
 2895 be to ignore domain differences, and simply merge the training data from the source and
 2896 target domains. There are several other baseline approaches to dealing with this sce-
 2897 nario (Daumé III, 2007):

2898 **Interpolation.** Train a classifier for each domain, and combine their predictions, e.g.,

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

2899 where Ψ_s and Ψ_t are the scoring functions from the source and target domain clas-
 2900 sifiers respectively, and λ_s is the interpolation weight.

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

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

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

2901 **Priors.** Train a classifier on the source domain data, and use its weights as a prior distri-
 2902 bution on the weights of the classifier for the target domain data. This is equivalent
 2903 to regularizing the target domain weights towards the weights of the source domain
 2904 classifier (Chelba and Acero, 2006),

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

2905 where $\ell^{(i)}$ is the prediction loss on instance i , and λ is the regularization weight.

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

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

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

5.4.2 Unsupervised domain adaptation

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

Linear projection

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

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

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

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

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

2926 The projection matrix \mathbf{U} can be learned in a number of different ways, but many ap-
 2927 proaches focus on compressing and reconstructing the base features (Ando and Zhang,
 2928 2005). For example, we can define a set of **pivot features**, which are typically chosen be-
 2929 cause they appear in both domains: in the case of review documents, pivot features might
 2930 include evaluative adjectives like *outstanding* and *disappointing* (Blitzer et al., 2007). For
 2931 each pivot feature j , we define an auxiliary problem of predicting whether the feature is
 2932 present in each example, using the remaining base features. Let ϕ_j denote the weights of
 2933 this classifier, and us horizontally concatenate the weights for each of the N_p pivot features
 2934 into a matrix $\Phi = [\phi_1, \phi_2, \dots, \phi_{N_p}]$.

2935 We then perform truncated singular value decomposition on Φ , as described in § 5.2.1,
 2936 obtaining $\Phi \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top$. The rows of the matrix \mathbf{U} summarize information about each base
 2937 feature: indeed, the truncated singular value decomposition identifies a low-dimension
 2938 basis for the weight matrix Φ , which in turn links base features to pivot features. Sup-
 2939 pose that a base feature *reliable* occurs only in the target domain of appliance reviews.
 2940 Nonetheless, it will have a positive weight towards some pivot features (e.g., *outstanding*,
 2941 *recommended*), and a negative weight towards others (e.g., *worthless*, *unpleasant*). A base
 2942 feature such as *watchable* might have the same associations with the pivot features, and
 2943 therefore, $\mathbf{u}_{\text{reliable}} \approx \mathbf{u}_{\text{watchable}}$. The matrix \mathbf{U} can thus project the base features into a
 2944 space in which this information is shared.

2945 Non-linear projection

2946 Non-linear transformations of the base features can be accomplished by implementing
 2947 the transformation function as a deep neural network, which is trained from an auxiliary
 2948 objective.

2949 **Denoising objectives** One possibility is to train a projection function to reconstruct a
 2950 corrupted version of the original input. The original input can be corrupted in various
 2951 ways: by the addition of random noise (Glorot et al., 2011; Chen et al., 2012), or by the
 2952 deletion of features (Chen et al., 2012; Yang and Eisenstein, 2015). Denoising objectives
 2953 share many properties of the linear projection method described above: they enable the
 2954 projection function to be trained on large amounts of unlabeled data from the target do-
 2955 main, and allow information to be shared across the feature space, thereby reducing sen-
 2956 sitivity to rare and domain-specific features.

2957 **Adversarial objectives** The ultimate goal is for the transformed representations $g(x^{(i)})$
 2958 to be domain-general. This can be made an explicit optimization criterion by comput-
 2959 ing the similarity of transformed instances both within and between domains (Tzeng
 2960 et al., 2015), or by formulating an auxiliary classification task, in which the domain it-
 2961 self is treated as a label (Ganin et al., 2016). This setting is **adversarial**, because we want

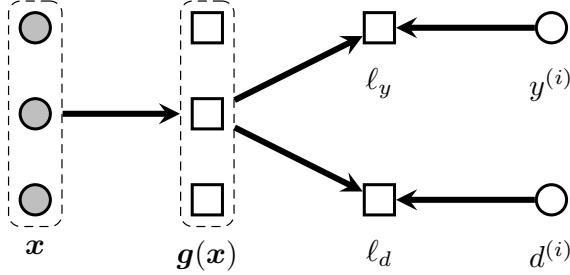


Figure 5.4: A schematic view of adversarial domain adaptation. The loss ℓ_y is computed only for instances from the source domain, where labels $y^{(i)}$ are available.

2962 to learn a representation that makes this classifier perform poorly. At the same time, we
 2963 want $g(x^{(i)})$ to enable accurate predictions of the labels $y^{(i)}$.

2964 To formalize this idea, let $d^{(i)}$ represent the domain of instance i , and let $\ell_d(g(x^{(i)}), d^{(i)}; \theta_d)$
 2965 represent the loss of a classifier (typically a deep neural network) trained to predict $d^{(i)}$
 2966 from the transformed representation $g(x^{(i)})$, using parameters θ_d . Analogously, let $\ell_y(g(x^{(i)}), y^{(i)}; \theta_y)$
 2967 represent the loss of a classifier trained to predict the label $y^{(i)}$ from $g(x^{(i)})$, using param-
 2968 eters θ_y . The transformation g can then be trained from two criteria: it should yield accu-
 2969 rate predictions of the labels $y^{(i)}$, while making *inaccurate* predictions of the domains $d^{(i)}$.
 2970 This can be formulated as a joint optimization problem,

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

2971 where N_ℓ is the number of labeled instances and N_u is the number of unlabeled instances,
 2972 with the labeled instances appearing first in the dataset. This setup is shown in Figure 5.4.
 2973 The loss can be optimized by stochastic gradient descent, jointly training the parameters
 2974 of the non-linear transformation θ_g , and the parameters of the prediction models θ_d and
 2975 θ_y .

2976 5.5 *Other approaches to learning with latent variables

2977 Expectation-maximization provides a general approach to learning with latent variables,
 2978 but it has limitations. One is the sensitivity to initialization; in practical applications,
 2979 considerable attention may need to be devoted to finding a good initialization. A second
 2980 issue is that EM tends to be easiest to apply in cases where the latent variables have a clear
 2981 decomposition (in the cases we have considered, they decompose across the instances).
 2982 For these reasons, it is worth briefly considering some alternatives to EM.

2983 **5.5.1 Sampling**

2984 In EM clustering, there is a distribution $\mathbf{q}^{(i)}$ for the missing data related to each instance.
 2985 The M-step consists of updating the parameters of this distribution. An alternative is to
 2986 draw samples of the latent variables. If the sampling distribution is designed correctly,
 2987 this procedure will eventually converge to drawing samples from the true posterior over
 2988 the missing data, $p(z^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$. For example, in the case of clustering, the missing
 2989 data $\mathbf{z}^{(1:N_z)}$ is the set of cluster memberships, $\mathbf{y}^{(1:N)}$, so we draw samples from the pos-
 2990 terior distribution over clusterings of the data. If a single clustering is required, we can
 2991 select the one with the highest conditional likelihood, $\hat{\mathbf{z}} = \text{argmax}_{\mathbf{z}} p(\mathbf{z}^{(1:N_z)} | \mathbf{x}^{(1:N_x)})$.

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

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

where $\mathbf{z}^{(-n)}$ indicates $\{\mathbf{z} \setminus z^{(n)}\}$, the set of all latent variables except for $z^{(n)}$. Repeatedly drawing samples over all latent variables constructs a Markov chain that 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,

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

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

2992 In this case, the sampling distribution does not depend on the other instances: the poste-
 2993 rior distribution over each $z^{(i)}$ can be computed from $\mathbf{x}^{(i)}$ and the parameters given the
 2994 parameters $\boldsymbol{\phi}$ and $\boldsymbol{\mu}$.

2995 In sampling algorithms, there are several choices for how to deal with the parameters.
 2996 One possibility is to sample them too. To do this, we must add them to the generative
 2997 story, by introducing a prior distribution. For the multinomial and categorical parameters
 2998 in the EM clustering model, the **Dirichlet distribution** is a typical choice, since it defines
 2999 a probability on exactly the set of vectors that can be parameters: vectors that sum to one
 3000 and include only non-negative numbers.¹⁰

3001 To incorporate this prior, the generative model must be augmented to indicate that
 3002 each $\boldsymbol{\phi}_z \sim \text{Dirichlet}(\boldsymbol{\alpha}_\phi)$, and $\boldsymbol{\mu} \sim \text{Dirichlet}(\boldsymbol{\alpha}_\mu)$. The hyperparameters $\boldsymbol{\alpha}$ are typically set
 3003 to a constant vector $\boldsymbol{\alpha} = [\alpha, \alpha, \dots, \alpha]$. When α is large, the Dirichlet distribution tends to

¹⁰If $\sum_i^K \theta_i = 1$ and $\theta_i \geq 0$ for all i , then $\boldsymbol{\theta}$ is said to be on the $K - 1$ simplex. A Dirichlet distribution with

3004 generate vectors that are nearly uniform; when α is small, it tends to generate vectors that
 3005 assign most of their probability mass to a few entries. Given prior distributions over ϕ
 3006 and μ , we can now include them in Gibbs sampling, drawing values for these parameters
 3007 from posterior distributions that are conditioned on the other variables in the model.

3008 Unfortunately, sampling ϕ and μ usually leads to slow “mixing”, meaning that adja-
 3009 cent samples tend to be similar, so that a large number of samples is required to explore
 3010 the space of random variables. The reason is that the sampling distributions for the pa-
 3011 rameters are tightly constrained by the cluster memberships $y^{(i)}$, which in turn are tightly
 3012 constrained by the parameters. There are two solutions that are frequently employed:

- 3013 • **Empirical Bayesian** methods maintain ϕ and μ as parameters rather than latent
 3014 variables. They still employ sampling in the E-step of the EM algorithm, but they
 3015 update the parameters using expected counts that are computed from the samples
 3016 rather than from parametric distributions. This EM-MCMC hybrid is also known
 3017 as Monte Carlo Expectation Maximization (MCEM; Wei and Tanner, 1990), and is
 3018 well-suited for cases in which it is difficult to compute $q^{(i)}$ directly.
- 3019 • In **collapsed Gibbs sampling**, we analytically integrate ϕ and μ out of the model.
 3020 The cluster memberships $y^{(i)}$ are the only remaining latent variable; we sample them
 3021 from the compound distribution,

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

3022 For multinomial and Dirichlet distributions, this integral can be computed in closed
 3023 form.

3024 MCMC algorithms are guaranteed to converge to the true posterior distribution over
 3025 the latent variables, but there is no way to know how long this will take. In practice, the
 3026 rate of convergence depends on initialization, just as expectation-maximization depends
 3027 on initialization to avoid local optima. Thus, while Gibbs Sampling and other MCMC
 3028 algorithms provide a powerful and flexible array of techniques for statistical inference in
 3029 latent variable models, they are not a panacea for the problems experienced by EM.

parameter $\alpha \in \mathbb{R}_+^K$ has support over the $K - 1$ simplex,

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

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

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

3030 **5.5.2 Spectral learning**

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

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

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

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

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

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

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

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

3031 which is similar to the eigendecomposition $\mathbf{C} = \mathbf{Q} \boldsymbol{\Lambda} \mathbf{Q}^\top$. This suggests that simply by
 3032 finding the eigenvectors and eigenvalues of \mathbf{C} , we could obtain the parameters ϕ and μ ,
 3033 and this is what motivates the name **spectral learning**.

3034 While moment-matching and eigendecomposition are similar in form, they impose
 3035 different constraints on the solutions: eigendecomposition requires orthonormality, so
 3036 that $\mathbf{Q}\mathbf{Q}^\top = \mathbb{I}$; in estimating the parameters of a text clustering model, we require that μ
 3037 and the columns of Φ are probability vectors. Spectral learning algorithms must therefore
 3038 include a procedure for converting the solution into vectors that are non-negative and
 3039 sum to one. One approach is to replace eigendecomposition (or the related singular value
 3040 decomposition) with non-negative matrix factorization (Xu et al., 2003), which guarantees
 3041 that the solutions are non-negative (Arora et al., 2013).

3042 After obtaining the parameters ϕ and μ , the distribution over clusters can be com-
 3043 puted from Bayes' rule:

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

3044 Spectral learning yields provably good solutions without regard to initialization, and can
 3045 be quite fast in practice. However, it is more difficult to apply to a broad family of genera-
 3046 tive models than EM and Gibbs Sampling. For more on applying spectral learning across
 3047 a range of latent variable models, see Anandkumar et al. (2014).

3048 Additional resources

3049 There are a number of other learning paradigms that deviate from supervised learning.

- 3050 • **Active learning:** the learner selects unlabeled instances and requests annotations (Set-
 3051 tles, 2012).
- 3052 • **Multiple instance learning:** labels are applied to bags of instances, with a positive
 3053 label applied if at least one instance in the bag meets the criterion (Dietterich et al.,
 3054 1997; Maron and Lozano-Pérez, 1998).
- 3055 • **Constraint-driven learning:** supervision is provided in the form of explicit con-
 3056 straints on the learner (Chang et al., 2007; Ganchev et al., 2010).
- 3057 • **Distant supervision:** noisy labels are generated from an external resource (Mintz
 3058 et al., 2009, also see § 17.2.3).
- 3059 • **Multitask learning:** the learner induces a representation that can be used to solve
 3060 multiple classification tasks (Collobert et al., 2011).
- 3061 • **Transfer learning:** the learner must solve a classification task that differs from the
 3062 labeled data (Pan and Yang, 2010).

3063 Expectation-maximization was introduced by Dempster et al. (1977), and is discussed
 3064 in more detail by Murphy (2012). Like most machine learning treatments, Murphy focuses
 3065 on continuous observations and Gaussian likelihoods, rather than the discrete observa-
 3066 tions typically encountered in natural language processing. Murphy (2012) also includes
 3067 an excellent chapter on MCMC; for a textbook-length treatment, see Robert and Casella
 3068 (2013). For still more on Bayesian latent variable models, see Barber (2012), and for ap-
 3069 plications of Bayesian models to natural language processing, see Cohen (2016). Surveys
 3070 are available for semi-supervised learning (Zhu and Goldberg, 2009) and domain adapta-
 3071 tion (Søgaard, 2013), although both pre-date the current wave of interest in deep learning.

3072 Exercises

- 3073 1. Derive the expectation maximization update for the parameter μ in the EM cluster-
 3074 ing model.

- 3075 2. Derive the E-step and M-step updates for the following generative model. You may
 3076 assume that the labels $y^{(i)}$ are observed, but $z_m^{(i)}$ is not.

- 3077 • For each instance i ,

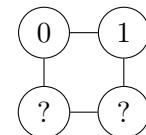
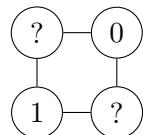
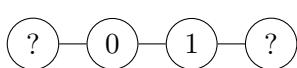
- 3078 – Draw label $y^{(i)} \sim \text{Categorical}(\mu)$
- 3079 – For each token $m \in \{1, 2, \dots, M^{(i)}\}$

3080 * Draw $z_m^{(i)} \sim \text{Categorical}(\pi)$

3081 * If $z_m^{(i)} = 0$, draw the current token from a label-specific distribution,
 3082 $w_m^{(i)} \sim \phi_{y^{(i)}}$

3083 * If $z_m^{(i)} = 1$, draw the current token from a document-specific distribu-
 3084 tion, $w_m^{(i)} \sim \nu^{(i)}$

- 3085 3. Using the iterative updates in Equations 5.34-5.36, compute the outcome of the label
 3086 propagation algorithm for the following examples.



3087 The value inside the node indicates the label, $y^{(i)} \in \{0, 1\}$, with $y^{(i)} = ?$ for unlabeled
 3088 nodes. The presence of an edge between two nodes indicates $w_{i,j} = 1$, and the
 3089 absence of an edge indicates $w_{i,j} = 0$. For the third example, you need only compute
 3090 the first three iterations, and then you can guess at the solution in the limit.

- 3091 4. Use expectation-maximization clustering to train a word-sense induction system,
 3092 applied to the word *say*.

- 3093 • Import NLTK, run NLTK.DOWNLOAD() and select SEMCOR. Import SEMCOR
 3094 from NLTK.CORPUS.
- 3095 • The command SEMCOR.TAGGED_SENTENCES(TAG='SENSE') returns an itera-
 3096 tor over sense-tagged sentences in the corpus. Each sentence can be viewed
 3097 as an iterator over TREE objects. For TREE objects that are sense-annotated
 3098 words, you can access the annotation as TREE.LABEL(), and the word itself with
 3099 TREE.LEAVES(). So SEMCOR.TAGGED_SENTENCES(TAG='SENSE')[0][2].LABEL()
 3100 would return the sense annotation of the third word in the first sentence.
- 3101 • Extract all sentences containing the senses SAY.V.01 and SAY.V.02.
- 3102 • Build bag-of-words vectors $x^{(i)}$, containing the counts of other words in those
 3103 sentences, including all words that occur in at least two sentences.
- 3104 • Implement and run expectation-maximization clustering on the merged data.

- 3105 • Compute the frequency with which each cluster includes instances of SAY.V.01
 3106 and SAY.V.02.

3107 In the remaining exercises, you will try out some approaches for semisupervised learn-
 3108 ing and domain adaptation. You will need datasets in multiple domains. You can obtain
 3109 product reviews in multiple domains here: [https://www.cs.jhu.edu/~mdredze/](https://www.cs.jhu.edu/~mdredze/datasets/sentiment/processed_acl.tar.gz)
 3110 datasets/sentiment/processed_acl.tar.gz. Choose a source and target domain,
 3111 e.g. dvds and books, and divide the data for the target domain into training and test sets
 3112 of equal size.

3113 5. First, quantify the cost of cross-domain transfer.

- 3114 • Train a logistic regression classifier on the source domain training set, and eval-
 3115 uate it on the target domain test set.
 3116 • Train a logistic regression classifier on the target domain training set, and eval-
 3117 uate it on the target domain test set. This is the “direct transfer” baseline.

3118 Compute the difference in accuracy, which is a measure of the transfer loss across
 3119 domains.

3120 6. Next, apply the **label propagation** algorithm from § 5.3.2.

3121 As a baseline, using only 5% of the target domain training set, train a classifier, and
 3122 compute its accuracy on the target domain test set.

3123 Next, apply label propagation:

- 3124 • Compute the label matrix \mathbf{Q}_L for the labeled data (5% of the target domain
 3125 training set), with each row equal to an indicator vector for the label (positive
 3126 or negative).
 3127 • Iterate through the target domain instances, including both test and training
 3128 data. At each instance i , compute all w_{ij} , using Equation 5.32, with $\alpha = 0.01$.
 3129 Use these values to fill in column i of the transition matrix \mathbf{T} , setting all but the
 3130 ten largest values to zero for each column i . Be sure to normalize the column
 3131 so that the remaining values sum to one. You may need to use a sparse matrix
 3132 for this to fit into memory.
 3133 • Apply the iterative updates from Equations 5.34–5.36 to compute the outcome
 3134 of the label propagation algorithm for the unlabeled examples.

3135 Select the test set instances from \mathbf{Q}_U , and compute the accuracy of this method.
 3136 Compare with the supervised classifier trained only on the 5% sample of the target
 3137 domain training set.

- 3138 7. Using only 5% of the target domain training data (and all of the source domain train-
3139 ing data), implement one of the supervised domain adaptation baselines in § 5.4.1.
3140 See if this improves on the “direct transfer” baseline from the previous problem
- 3141 8. Implement EASYADAPT (§ 5.4.1), again using 5% of the target domain training data
3142 and all of the source domain data.
- 3143 9. Now try unsupervised domain adaptation, using the “linear projection” method
3144 described in § 5.4.2. Specifically:
- 3145 • Identify 500 pivot features as the words with the highest frequency in the (com-
3146 plete) training data for the source and target domains. Specifically, let x_i^d be the
3147 count of the word i in domain d : choose the 500 words with the largest values
3148 of $\min(x_i^{\text{source}}, x_i^{\text{target}})$.
- 3149 • Train a classifier to predict each pivot feature from the remaining words in the
3150 document.
- 3151 • Arrange the features of these classifiers into a matrix Φ , and perform truncated
3152 singular value decomposition, with $k = 20$
- 3153 • Train a classifier from the source domain data, using the combined features
3154 $\mathbf{x}^{(i)} \oplus \mathbf{U}^\top \mathbf{x}^{(i)}$ — these include the original bag-of-words features, plus the pro-
3155 jected features.
- 3156 • Apply this classifier to the target domain test set, and compute the accuracy.

3157

Part II

3158

Sequences and trees

3159

Chapter 6

3160

Language models

3161 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,

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

3165 Why would you want to compute the probability of a word sequence? In many applications, the goal is to produce word sequences as output:

- 3167 • In **machine translation** (chapter 18), we convert from text in a source language to text in a target language.
- 3169 • In **speech recognition**, we convert from audio signal to text.
- 3170 • In **summarization** (§ 16.3.4; § 19.2), we convert from long texts into short texts.
- 3171 • In **dialogue systems** (§ 19.3), we convert from the user's input (and perhaps an external knowledge base) into a text response.

3173 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.

3177 (6.1) El cafe negro me gusta mucho.

3178 Here is a literal word-for-word translation (a **gloss**):

3179 (6.2) The coffee black me pleases much.

3180 A good language model of English will tell us that the probability of this translation is
 3181 low, in comparison with more grammatical alternatives,

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

3182 How can we use this fact? Warren Weaver, one of the early leaders in machine trans-
 3183 lation, viewed it as a problem of breaking a secret code (Weaver, 1955):

3184 When I look at an article in Russian, I say: 'This is really written in English,
 3185 but it has been coded in some strange symbols. I will now proceed to decode.'

3186 This observation motivates a generative model (like Naïve Bayes):

3187 • The English sentence $w^{(e)}$ is generated from a **language model**, $p_e(w^{(e)})$.

3188 • The Spanish sentence $w^{(s)}$ is then generated from a **translation model**, $p_{s|e}(w^{(s)} | w^{(e)})$.

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

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

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

3189 This is sometimes called the **noisy channel model**, because it envisions English text
 3190 turning into Spanish by passing through a noisy channel, $p_{s|e}$. What is the advantage of
 3191 modeling translation this way, as opposed to modeling $p_{e|s}$ directly? The crucial point is
 3192 that the two distributions $p_{s|e}$ (the translation model) and p_e (the language model) can be
 3193 estimated from separate data. The translation model requires examples of correct trans-
 3194 lations, but the language model requires only text in English. Such monolingual data is
 3195 much more widely available. Furthermore, once estimated, the language model p_e can
 3196 be reused in any application that involves generating English text, including translation
 3197 from other languages.

3198 6.1 *N*-gram language models

A simple approach to computing the probability of a sequence of tokens is to use a **relative frequency estimate**. Consider the quote, attributed to Picasso, "computers are useless, they can only give you answers." One way to estimate the probability of this sentence is,

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

3199 This estimator is **unbiased**: in the theoretical limit of infinite data, the estimate will
 3200 be correct. But in practice, we are asking for accurate counts over an infinite number of
 3201 events, since sequences of words can be arbitrarily long. Even with an aggressive upper
 3202 bound of, say, $M = 20$ tokens in the sequence, the number of possible sequences is V^{20} ,
 3203 where $V = |\mathcal{V}|$. A small vocabulary for English would have $V = 10^5$, so there are 10^{100}
 3204 possible sequences. Clearly, this estimator is very data-hungry, and suffers from high vari-
 3205 ance: even grammatical sentences will have probability zero if they have not occurred in
 3206 the training data.¹ We therefore need to introduce bias to have a chance of making reli-
 3207 able estimates from finite training data. The language models that follow in this chapter
 3208 introduce bias in various ways.

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

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

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

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

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

3209 We haven't made any approximations yet, and we could have just as well applied the
 3210 chain rule in reverse order,

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

3211 or in any other order. But this means that we also haven't really made any progress:
 3212 to compute the conditional probability $p(w_M | w_{M-1}, w_{M-2}, \dots, w_1)$, we would need to
 3213 model V^{M-1} contexts. Such a distribution cannot be estimated from any realistic sample
 3214 of text.

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

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

¹Chomsky famously argued that this is evidence against the very concept of probabilistic language models: no such model could distinguish the grammatical sentence *colorless green ideas sleep furiously* from the ungrammatical permutation *furiously sleep ideas green colorless*.

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

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

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

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

3215 This model requires estimating and storing the probability of only V^n events, which is
 3216 exponential in the order of the n -gram, and not V^M , which is exponential in the length of
 3217 the sentence. The n -gram probabilities can be computed by relative frequency estimation,

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

3218 The hyperparameter n controls the size of the context used in each conditional proba-
 3219 bility. If this is misspecified, the language model will perform poorly. Let's consider the
 3220 potential problems concretely.

3221 **When n is too small.** Consider the following sentences:

3222 (6.3) **Gorillas** always like to groom **their** friends.

3223 (6.4) The **computer** that's on the 3rd floor of our office building **crashed**.

3224 In each example, the words written in bold depend on each other: the likelihood
 3225 of *their* depends on knowing that *gorillas* is plural, and the likelihood of *crashed* de-
 3226 pends on knowing that the subject is a *computer*. If the n -grams are not big enough
 3227 to capture this context, then the resulting language model would offer probabili-
 3228 ties that are too low for these sentences, and too high for sentences that fail basic
 3229 linguistic tests like number agreement.

3230 **When n is too big.** In this case, it is hard to get good estimates of the n -gram parameters from
 3231 our dataset, because of data sparsity. To handle the *gorilla* example, it is necessary to
 3232 model 6-grams, which means accounting for V^6 events. Under a very small vocab-
 3233 uary of $V = 10^4$, this means estimating the probability of 10^{24} distinct events.

3234 These two problems point to another **bias-variance tradeoff** (see § 2.2.4). A small n -
 3235 gram size introduces high bias, and a large n -gram size introduces high variance. We
 3236 can even have both problems at the same time! Language is full of long-range dependen-
 3237 cies that we cannot capture because n is too small; at the same time, language datasets
 3238 are full of rare phenomena, whose probabilities we fail to estimate accurately because n
 3239 is too large. One solution is to try to keep n large, while still making low-variance esti-
 3240 mates of the underlying parameters. To do this, we will introduce a different sort of bias:
 3241 **smoothing**.

3242 6.2 Smoothing and discounting

3243 Limited data is a persistent problem in estimating language models. In § 6.1, we pre-
 3244 sented n -grams as a partial solution. Bit sparse data can be a problem even for low-order
 3245 n -grams; at the same time, many linguistic phenomena, like subject-verb agreement, can-
 3246 not be incorporated into language models without high-order n -grams. It is therefore
 3247 necessary to add additional inductive biases to n -gram language models. This section
 3248 covers some of the most intuitive and common approaches, but there are many more (see
 3249 Chen and Goodman, 1999).

3250 6.2.1 Smoothing

3251 A major concern in language modeling is to avoid the situation $p(w) = 0$, which could
 3252 arise as a result of a single unseen n-gram. A similar problem arose in Naïve Bayes, and
 3253 the solution was **smoothing**: adding imaginary “pseudo” counts. The same idea can be
 3254 applied to n -gram language models, as shown here in the bigram case,

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

3255 This basic framework is called **Lidstone smoothing**, but special cases have other names:

- 3256 • **Laplace smoothing** corresponds to the case $\alpha = 1$.
- 3257 • **Jeffreys-Perks law** corresponds to the case $\alpha = 0.5$, which works well in practice
 3258 and benefits from some theoretical justification (Manning and Schütze, 1999).

3259 To ensure that the probabilities are properly normalized, anything that we add to the
 3260 numerator (α) must also appear in the denominator ($V\alpha$). This idea is reflected in the
 3261 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,

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

3262 6.2.2 Discounting and backoff

3263 Discounting “borrows” probability mass from observed n -grams and redistributes it. In
 3264 Lidstone smoothing, the borrowing is done by increasing the denominator of the relative
 3265 frequency estimates. The borrowed probability mass is then redistributed by increasing
 3266 the numerator for all n -grams. Another approach would be to borrow the same amount
 3267 of probability mass from all observed n -grams, and redistribute it among only the unob-
 3268 served n -grams. This is called **absolute discounting**. For example, suppose we set an
 3269 absolute discount $d = 0.1$ in a bigram model, and then redistribute this probability mass
 3270 equally over the unseen words. The resulting probabilities are shown in Table 6.1.

Discounting reserves some probability mass from the observed data, and we need not redistribute this probability mass equally. Instead, we can **backoff** to a lower-order language model: if you have trigrams, use trigrams; if you don’t have trigrams, use bigrams; if you don’t even have bigrams, use unigrams. This is called **Katz backoff**. In the simple

case of backing off from bigrams to unigrams, the bigram probabilities are,

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

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

3271 The term $\alpha(j)$ indicates the amount of probability mass that has been discounted for
 3272 context j . This probability mass is then divided across all the unseen events, $\{i' : c(i', j) =$
 3273 $0\}$, proportional to the unigram probability of each word i' . The discount parameter d can
 3274 be optimized to maximize performance (typically held-out log-likelihood) on a develop-
 3275 ment set.

3276 6.2.3 *Interpolation

3277 Backoff is one way to combine different order n -gram models. An alternative approach
 3278 is **interpolation**: setting the probability of a word in context to a weighted sum of its
 3279 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,

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

3280 In this equation, p_n^* is the unsmoothed empirical probability given by an n -gram lan-
 3281 guage model, and λ_n is the weight assigned to this model. To ensure that the interpolated
 3282 $p(w)$ is still a valid probability distribution, the values of λ must obey the constraint,
 3283 $\sum_{n=1}^{n_{\max}} \lambda_n = 1$. But how to find the specific values?

3284 An elegant solution is **expectation-maximization**. Recall from chapter 5 that we can
 3285 think about EM as learning with *missing data*: we just need to choose missing data such
 3286 that learning would be easy if it weren't missing. What's missing in this case? Think of
 3287 each word w_m as drawn from an n -gram of unknown size, $z_m \in \{1 \dots n_{\max}\}$. This z_m is
 3288 the missing data that we are looking for. Therefore, the application of EM to this problem
 3289 involves the following **generative model**:

3290 **for** Each token $w_m, m = 1, 2, \dots, M$ **do**:
 3291 draw the n -gram size $z_m \sim \text{Categorical}(\lambda)$;
 3292 draw $w_m \sim p_{z_m}^*(w_m | w_{m-1}, \dots, w_{m-z_m})$.

If the missing data $\{Z_m\}$ were known, then λ could be estimated as the relative frequency,

$$\lambda_z = \frac{\text{count}(Z_m = z)}{M} \quad [6.17]$$

$$\propto \sum_{m=1}^M \delta(Z_m = z). \quad [6.18]$$

But since we do not know the values of the latent variables Z_m , we impute a distribution q_m in the E-step, which represents the degree of belief that word token w_m was generated from a n -gram of order z_m ,

$$q_m(z) \triangleq \Pr(Z_m = z \mid \mathbf{w}_{1:m}; \lambda) \quad [6.19]$$

$$= \frac{p(w_m \mid \mathbf{w}_{1:m-1}, Z_m = z) \times p(z)}{\sum_{z'} p(w_m \mid \mathbf{w}_{1:m-1}, Z_m = z') \times p(z')} \quad [6.20]$$

$$\propto p_z^*(w_m \mid \mathbf{w}_{1:m-1}) \times \lambda_z. \quad [6.21]$$

In the M-step, λ is computed by summing the expected counts under q ,

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

3294 A solution is obtained by iterating between updates to q and λ . The complete algorithm
 3295 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  $\lambda$ 
  
```

3296 **6.2.4 *Kneser-Ney smoothing**

3297 **Kneser-Ney smoothing** is based on absolute discounting, but it redistributes the result-
 3298 ing probability mass in a different way from Katz backoff. Empirical evidence points
 3299 to Kneser-Ney smoothing as the state-of-art for n -gram language modeling (Goodman,
 3300 2001). To motivate Kneser-Ney smoothing, consider the example: *I recently visited ...*
 3301 Which of the following is more likely?

3302 • *Francisco*

3303 • *Duluth*

3304 Now suppose that both bigrams *visited Duluth* and *visited Francisco* are unobserved in
 3305 the training data, and furthermore, the unigram probability $p_1^*(\text{Francisco})$ is greater than
 3306 $p^*(\text{Duluth})$. Nonetheless we would still guess that $p(\text{visited Duluth}) > p(\text{visited Francisco})$,
 3307 because *Duluth* is a more “versatile” word: it can occur in many contexts, while *Francisco*
 3308 usually occurs in a single context, following the word *San*. This notion of versatility is the
 3309 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

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

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

3310 Probability mass using absolute discounting d , which is taken from all unobserved
 3311 n -grams. The total amount of discounting in context u is $d \times |w : \text{count}(w, u) > 0|$, and
 3312 we divide this probability mass among the unseen n -grams. To account for versatility,
 3313 we define the *continuation probability* $p_{\text{continuation}}(w)$ as proportional to the number of ob-
 3314 served contexts in which w appears. The numerator of the continuation probability is the
 3315 number of contexts u in which w appears; the denominator normalizes the probability by
 3316 summing the same quantity over all words w' . The coefficient $\alpha(u)$ is set to ensure that
 3317 the probability distribution $p_{KN}(w | u)$ sums to one over the vocabulary w .

3318 The idea of modeling versatility by counting contexts may seem heuristic, but there is
 3319 an elegant theoretical justification from Bayesian nonparametrics (Teh, 2006). Kneser-Ney
 3320 smoothing on n -grams was the dominant language modeling technique before the arrival
 3321 of neural language models.

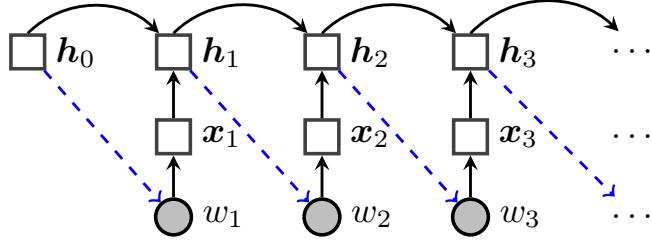


Figure 6.1: The recurrent neural network language model, viewed as an “unrolled” computation graph. Solid lines indicate direct computation, dotted blue lines indicate probabilistic dependencies, circles indicate random variables, and squares indicate computation nodes.

3322 6.3 Recurrent neural network language models

3323 N -gram language models have been largely supplanted by neural networks. These mod-
 3324 els do not make the n -gram assumption of restricted context; indeed, they can incorpo-
 3325 rate arbitrarily distant contextual information, while remaining computationally and statis-
 3326 tically tractable.

3327 The first insight behind neural language models is to treat word prediction as a *dis-
 3328 criminative* learning task.² The goal is to compute the probability $p(w | u)$, where $w \in \mathcal{V}$ is
 3329 a word, and u is the context, which depends on the previous words. Rather than directly
 3330 estimating the word probabilities from (smoothed) relative frequencies, we can treat
 3331 language modeling as a machine learning problem, and estimate parameters that maxi-
 3332 mize the log conditional probability of a corpus.

3333 The second insight is to reparametrize the probability distribution $p(w | u)$ as a func-
 3334 tion of two dense K -dimensional numerical vectors, $\beta_w \in \mathbb{R}^K$, and $v_u \in \mathbb{R}^K$,

$$p(w | u) = \frac{\exp(\beta_w \cdot v_u)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot v_u)}, \quad [6.25]$$

3335 where $\beta_w \cdot v_u$ represents a dot product. As usual, the denominator ensures that the prob-
 3336 ability distribution is properly normalized. This vector of probabilities is equivalent to
 3337 applying the **softmax** transformation (see § 3.1) to the vector of dot-products,

$$p(\cdot | u) = \text{SoftMax}([\beta_1 \cdot v_u, \beta_2 \cdot v_u, \dots, \beta_V \cdot v_u]). \quad [6.26]$$

The word vectors β_w are parameters of the model, and are estimated directly. The context vectors v_u can be computed in various ways, depending on the model. A simple

²This idea predates neural language models (e.g., Rosenfeld, 1996; Roark et al., 2007).

but effective neural language model can be built from a **recurrent neural network** (RNN; Mikolov et al., 2010). The basic idea is to recurrently update the context vectors while moving through the sequence. Let \mathbf{h}_m represent the contextual information at position m in the sequence. RNN language models are defined,

$$\mathbf{x}_m \triangleq \phi_{w_m} \quad [6.27]$$

$$\mathbf{h}_m = \text{RNN}(\mathbf{x}_m, \mathbf{h}_{m-1}) \quad [6.28]$$

$$p(w_{m+1} | w_1, w_2, \dots, w_m) = \frac{\exp(\beta_{w_{m+1}} \cdot \mathbf{h}_m)}{\sum_{w' \in \mathcal{V}} \exp(\beta_{w'} \cdot \mathbf{h}_m)}, \quad [6.29]$$

where ϕ is a matrix of **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),

$$\text{RNN}(\mathbf{x}_m, \mathbf{h}_{m-1}) \triangleq g(\Theta \mathbf{h}_{m-1} + \mathbf{x}_m), \quad [6.30]$$

where $\Theta \in \mathbb{R}^{K \times K}$ is the recurrence matrix and g is a non-linear transformation function, often defined as the elementwise hyperbolic tangent \tanh (see § 3.1).³ The \tanh acts as a **squashing function**, ensuring that each element of \mathbf{h}_m is constrained to the range $[-1, 1]$.

Although each w_m depends on only the context vector \mathbf{h}_{m-1} , this vector is in turn influenced by *all* previous tokens, w_1, w_2, \dots, w_{m-1} , through the recurrence operation: w_1 affects \mathbf{h}_1 , which affects \mathbf{h}_2 , and so on, until the information is propagated all the way to \mathbf{h}_{m-1} , and then on to w_m (see Figure 6.1). This is an important distinction from n -gram language models, where any information outside the n -word window is ignored. In principle, the RNN language model can handle long-range dependencies, such as number agreement over long spans of text — although it would be difficult to know where exactly in the vector \mathbf{h}_m this information is represented. The main limitation is that information is attenuated by repeated application of the squashing function g . **Long short-term memories** (LSTMs), described below, are a variant of RNNs that address this issue, using memory cells to propagate information through the sequence without applying nonlinearities (Hochreiter and Schmidhuber, 1997).

The denominator in Equation 6.29 is a computational bottleneck, because it involves a sum over the entire vocabulary. One solution is to use a **hierarchical softmax** function, which computes the sum more efficiently by organizing the vocabulary into a tree (Mikolov et al., 2011). Another strategy is to optimize an alternative metric, such as **noise-contrastive estimation** (Gutmann and Hyvärinen, 2012), which learns by distinguishing observed instances from artificial instances generated from a noise distribution (Mnih and Teh, 2012). Both of these strategies are described in § 14.5.3.

³In the original Elman network, the sigmoid function was used in place of \tanh . For an illuminating mathematical discussion of the advantages and disadvantages of various nonlinearities in recurrent neural networks, see the lecture notes from Cho (2015).

3364 **6.3.1 Backpropagation through time**

3365 The recurrent neural network language model has the following parameters:

- 3366 • $\phi_i \in \mathbb{R}^K$, the “input” word vectors (these are sometimes called **word embeddings**,
3367 since each word is embedded in a K -dimensional space; see chapter 14);
- 3368 • $\beta_i \in \mathbb{R}^K$, the “output” word vectors;
- 3369 • $\Theta \in \mathbb{R}^{K \times K}$, the recurrence operator;
- 3370 • \mathbf{h}_0 , the initial state.

3371 Each of these parameters can be estimated by formulating an objective function over the
3372 training corpus, $L(\mathbf{w})$, and then applying backpropagation to obtain gradients on the
3373 parameters from a minibatch of training examples (see § 3.3.1). Gradient-based updates
3374 can be computed from an online learning algorithm such as stochastic gradient descent
3375 (see § 2.6.2).

3376 The application of backpropagation to recurrent neural networks is known as **back-**
3377 **propagation through time**, because the gradients on units at time m depend in turn on the
3378 gradients of units at earlier times $n < m$. Let ℓ_{m+1} represent the negative log-likelihood
3379 of word $m + 1$,

$$\ell_{m+1} = -\log p(w_{m+1} | w_1, w_2, \dots, w_m). \quad [6.31]$$

We require the gradient of this loss with respect to each parameter, such as $\theta_{k,k'}$, an individual element in the recurrence matrix Θ . Since the loss depends on the parameters only through \mathbf{h}_m , we can apply the chain rule of differentiation,

$$\frac{\partial \ell_{m+1}}{\partial \theta_{k,k'}} = \frac{\partial \ell_{m+1}}{\partial \mathbf{h}_m} \frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}. \quad [6.32]$$

The vector \mathbf{h}_m depends on Θ in several ways. First, \mathbf{h}_m is computed by multiplying Θ by the previous state \mathbf{h}_{m-1} . But the previous state \mathbf{h}_{m-1} also depends on Θ :

$$\mathbf{h}_m = g(\mathbf{x}_m, \mathbf{h}_{m-1}) \quad [6.33]$$

$$\frac{\partial h_{m,k}}{\partial \theta_{k,k'}} = g'(\mathbf{x}_{m,k} + \boldsymbol{\theta}_k \cdot \mathbf{h}_{m-1})(h_{m-1,k'} + \boldsymbol{\theta}_k \cdot \frac{\partial \mathbf{h}_{m-1}}{\partial \theta_{k,k'}}), \quad [6.34]$$

3380 where g' is the local derivative of the nonlinear function g . The key point in this equation
3381 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
3382 so on, until reaching the initial state \mathbf{h}_0 .

3383 Each derivative $\frac{\partial \mathbf{h}_m}{\partial \theta_{k,k'}}$ will be reused many times: it appears in backpropagation from
3384 the loss ℓ_m , but also in all subsequent losses $\ell_{n>m}$. Neural network toolkits such as
3385 Torch (Collobert et al., 2011) and DyNet (Neubig et al., 2017) compute the necessary

3386 derivatives automatically, and cache them for future use. An important distinction from
3387 the feedforward neural networks considered in chapter 3 is that the size of the computa-
3388 tion graph is not fixed, but varies with the length of the input. This poses difficulties for
3389 toolkits that are designed around static computation graphs, such as TensorFlow (Abadi
3390 et al., 2016).⁴

3391 **6.3.2 Hyperparameters**

3392 The RNN language model has several hyperparameters that must be tuned to ensure good
3393 performance. The model capacity is controlled by the size of the word and context vectors
3394 K , which play a role that is somewhat analogous to the size of the n -gram context. For
3395 datasets that are large with respect to the vocabulary (i.e., there is a large token-to-type
3396 ratio), we can afford to estimate a model with a large K , which enables more subtle dis-
3397 tinctions between words and contexts. When the dataset is relatively small, then K must
3398 be smaller too, or else the model may “memorize” the training data, and fail to generalize.
3399 Unfortunately, this general advice has not yet been formalized into any concrete formula
3400 for choosing K , and trial-and-error is still necessary. Overfitting can also be prevented by
3401 **dropout**, which involves randomly setting some elements of the computation to zero (Sri-
3402 vastava et al., 2014), forcing the learner not to rely too much on any particular dimension
3403 of the word or context vectors. The dropout rate must also be tuned on development data.

3404 **6.3.3 Gated recurrent neural networks**

3405 In principle, recurrent neural networks can propagate information across infinitely long
3406 sequences. But in practice, repeated applications of the nonlinear recurrence function
3407 causes this information to be quickly attenuated. The same problem affects learning: back-
3408 propagation can lead to **vanishing gradients** that decay to zero, or **exploding gradients**
3409 that increase towards infinity (Bengio et al., 1994). The exploding gradient problem can
3410 be addressed by clipping gradients at some maximum value (Pascanu et al., 2013). The
3411 other issues must be addressed by altering the model itself.

3412 The **long short-term memory** (LSTM; Hochreiter and Schmidhuber, 1997) is a popular
3413 variant of RNNs that is more robust to these problems. This model augments the hidden
3414 state h_m with a **memory cell** c_m . The value of the memory cell at each time m is a gated
3415 sum of two quantities: its previous value c_{m-1} , and an “update” \tilde{c}_m , which is computed
3416 from the current input x_m and the previous hidden state h_{m-1} . The next state h_m is then
3417 computed from the memory cell. Because the memory cell is not passed through a non-
3418 linear squashing function during the update, it is possible for information to propagate
3419 through the network over long distances.

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

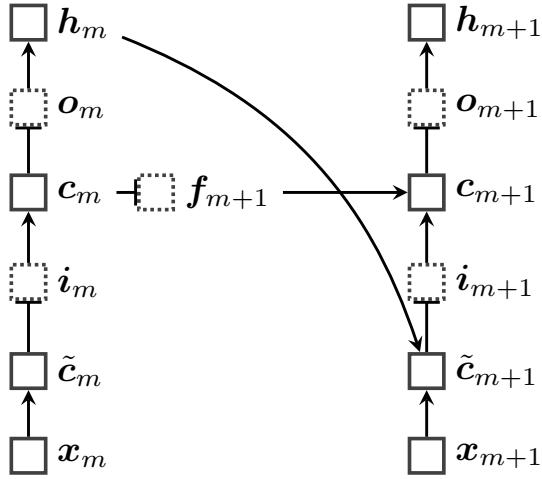


Figure 6.2: The long short-term memory (LSTM) architecture. Gates are shown in boxes with dotted edges. In an LSTM language model, each h_m would be used to predict the next word w_{m+1} .

The gates are functions of the input and previous hidden state. They are computed from elementwise sigmoid activations, $\sigma(x) = (1 + \exp(-x))^{-1}$, ensuring that their values will be in the range $[0, 1]$. They can therefore be viewed as soft, differentiable logic gates. The LSTM architecture is shown in Figure 6.2, and the complete update equations are:

$$f_{m+1} = \sigma(\Theta^{(h \rightarrow f)} h_m + \Theta^{(x \rightarrow f)} x_{m+1} + b_f) \quad \text{forget gate} \quad [6.35]$$

$$i_{m+1} = \sigma(\Theta^{(h \rightarrow i)} h_m + \Theta^{(x \rightarrow i)} x_{m+1} + b_i) \quad \text{input gate} \quad [6.36]$$

$$\tilde{c}_{m+1} = \tanh(\Theta^{(h \rightarrow c)} h_m + \Theta^{(x \rightarrow c)} x_{m+1}) \quad \text{update candidate} \quad [6.37]$$

$$c_{m+1} = f_{m+1} \odot c_m + i_{m+1} \odot \tilde{c}_{m+1} \quad \text{memory cell update} \quad [6.38]$$

$$o_{m+1} = \sigma(\Theta^{(h \rightarrow o)} h_m + \Theta^{(x \rightarrow o)} x_{m+1} + b_o) \quad \text{output gate} \quad [6.39]$$

$$h_{m+1} = o_{m+1} \odot \tanh(c_{m+1}) \quad \text{output.} \quad [6.40]$$

3420 The operator \odot is an elementwise (Hadamard) product. Each gate is controlled by a vec-
 3421 tor of weights, which parametrize the previous hidden state (e.g., $\Theta^{(h \rightarrow f)}$) and the current
 3422 input (e.g., $\Theta^{(x \rightarrow f)}$), plus a vector offset (e.g., b_f). The overall operation can be infor-
 3423 mally summarized as $(h_m, c_m) = \text{LSTM}(x_m, (h_{m-1}, c_{m-1}))$, with (h_m, c_m) representing
 3424 the LSTM state after reading token m .

3425 The LSTM outperforms standard recurrent neural networks across a wide range of
 3426 problems. It was first used for language modeling by Sundermeyer et al. (2012), but can
 3427 be applied more generally: the vector h_m can be treated as a complete representation of

3428 the input sequence up to position m , and can be used for any labeling task on a sequence
 3429 of tokens, as we will see in the next chapter.

3430 There are several LSTM variants, of which the Gated Recurrent Unit (Cho et al., 2014)
 3431 is one of the more well known. Many software packages implement a variety of RNN
 3432 architectures, so choosing between them is simple from a user’s perspective. Jozefowicz
 3433 et al. (2015) provide an empirical comparison of various modeling choices circa 2015.

3434 6.4 Evaluating language models

3435 Language modeling is not usually an application in itself: language models are typically
 3436 components of larger systems, and they would ideally be evaluated **extrinsically**. This
 3437 means evaluating whether the language model improves performance on the application
 3438 task, such as machine translation or speech recognition. But this is often hard to do, and
 3439 depends on details of the overall system which may be irrelevant to language modeling.
 3440 In contrast, **intrinsic evaluation** is task-neutral. Better performance on intrinsic metrics
 3441 may be expected to improve extrinsic metrics across a variety of tasks, but there is always
 3442 the risk of over-optimizing the intrinsic metric. This section discusses some intrinsic met-
 3443 rics, but keep in mind the importance of performing extrinsic evaluations to ensure that
 3444 intrinsic performance gains carry over to real applications.

3445 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,

$$\ell(\mathbf{w}) = \sum_{m=1}^M \log p(w_m | w_{m-1}, \dots, w_1), \quad [6.41]$$

3446 treating the entire held-out corpus as a single stream of tokens.

3447 Typically, unknown words are mapped to the $\langle \text{UNK} \rangle$ token. This means that we have
 3448 to estimate some probability for $\langle \text{UNK} \rangle$ on the training data. One way to do this is to fix
 3449 the vocabulary \mathcal{V} to the $V - 1$ words with the highest counts in the training data, and then
 3450 convert all other tokens to $\langle \text{UNK} \rangle$. Other strategies for dealing with out-of-vocabulary
 3451 terms are discussed in § 6.5.

3452 **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,

$$\text{Perplex}(\mathbf{w}) = 2^{-\frac{\ell(\mathbf{w})}{M}}, \quad [6.42]$$

3453 where M is the total number of tokens in the held-out corpus.

3454 Lower perplexities correspond to higher likelihoods, so lower scores are better on this
3455 metric — it is better to be less perplexed. Here are some special cases:

- 3456 • In the limit of a perfect language model, probability 1 is assigned to the held-out
3457 corpus, with $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 1} = 2^0 = 1$.
- 3458 • In the opposite limit, probability zero is assigned to the held-out corpus, which cor-
3459 responds to an infinite perplexity, $\text{Perplex}(\mathbf{w}) = 2^{-\frac{1}{M} \log_2 0} = 2^\infty = \infty$.
- 3460 • Assume a uniform, unigram model in which $p(w_i) = \frac{1}{V}$ for all words in the vocab-
3461 uary. Then,

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

3462 This is the “worst reasonable case” scenario, since you could build such a language
3463 model without even looking at the data.

3464 In practice, language models tend to give perplexities in the range between 1 and V .
3465 A small benchmark dataset is the **Penn Treebank**, which contains roughly a million to-
3466 kens; its vocabulary is limited to 10,000 words, with all other tokens mapped a special
3467 $\langle \text{UNK} \rangle$ symbol. On this dataset, a well-smoothed 5-gram model achieves a perplexity of
3468 141 (Mikolov and Zweig, Mikolov and Zweig), and an LSTM language model achieves
3469 perplexity of roughly 80 (Zaremba, Sutskever, and Vinyals, Zaremba et al.). Various en-
3470 hancements to the LSTM architecture can bring the perplexity below 60 (Merity et al.,
3471 2018). A larger-scale language modeling dataset is the 1B Word Benchmark (Chelba et al.,
3472 2013), which contains text from Wikipedia. On this dataset, a perplexities of around 25
3473 can be obtained by averaging together multiple LSTM language models (Jozefowicz et al.,
3474 2016).

3473 **6.5 Out-of-vocabulary words**

3474 So far, we have assumed a **closed-vocabulary** setting — the vocabulary \mathcal{V} is assumed to be
 3475 a finite set. In realistic application scenarios, this assumption may not hold. Consider, for
 3476 example, the problem of translating newspaper articles. The following sentence appeared
 3477 in a Reuters article on January 6, 2017:⁵

3478 The report said U.S. intelligence agencies believe Russian military intelligence,
 3479 the **GRU**, used intermediaries such as **WikiLeaks**, **DCLeaks.com** and the **Guccifer**
 3480 2.0 "persona" to release emails...

3481 Suppose that you trained a language model on the Gigaword corpus,⁶ which was released
 3482 in 2003. The bolded terms either did not exist at this date, or were not widely known; they
 3483 are unlikely to be in the vocabulary. The same problem can occur for a variety of other
 3484 terms: new technologies, previously unknown individuals, new words (e.g., *hashtag*), and
 3485 numbers.

3486 One solution is to simply mark all such terms with a special token, $\langle \text{UNK} \rangle$. While
 3487 training the language model, we decide in advance on the vocabulary (often the K most
 3488 common terms), and mark all other terms in the training data as $\langle \text{UNK} \rangle$. If we do not want
 3489 to determine the vocabulary size in advance, an alternative approach is to simply mark
 3490 the first occurrence of each word type as $\langle \text{UNK} \rangle$.

3491 But it is often better to make distinctions about the likelihood of various unknown words.
 3492 This is particularly important in languages that have rich morphological systems, with
 3493 many inflections for each word. For example, Portuguese is only moderately complex
 3494 from a morphological perspective, yet each verb has dozens of inflected forms (see Fig-
 3495 ure 4.3b). In such languages, there will be many word types that we do not encounter in a
 3496 corpus, which are nonetheless predictable from the morphological rules of the language.
 3497 To use a somewhat contrived English example, if *transfenestrate* is in the vocabulary, our
 3498 language model should assign a non-zero probability to the past tense *transfenestrated*,
 3499 even if it does not appear in the training data.

3500 One way to accomplish this is to supplement word-level language models with **character-**
 3501 **level language models**. Such models can use n -grams or RNNs, but with a fixed vocab-
 3502 uary equal to the set of ASCII or Unicode characters. For example, Ling et al. (2015)
 3503 propose an LSTM model over characters, and Kim (2014) employ a convolutional neural
 3504 network. A more linguistically motivated approach is to segment words into meaningful
 3505 subword units, known as **morphemes** (see chapter 9). For example, Botha and Blunsom

⁵Bayoumy, Y. and Strobel, W. (2017, January 6). U.S. intel report: Putin directed cyber campaign to help Trump. *Reuters*. Retrieved from <http://www.reuters.com/article/us-usa-russia-cyber-idUSKBN14Q1T8> on January 7, 2017.

⁶<https://catalog.ldc.upenn.edu/LDC2003T05>

3506 (2014) induce vector representations for morphemes, which they build into a log-bilinear
 3507 language model; Bhatia et al. (2016) incorporate morpheme vectors into an LSTM.

3508 Additional resources

3509 A variety of neural network architectures have been applied to language modeling. No-
 3510 table earlier non-recurrent architectures include the neural probabilistic language model (Ben-
 3511 gio et al., 2003) and the log-bilinear language model (Mnih and Hinton, 2007). Much more
 3512 detail on these models can be found in the text by Goodfellow et al. (2016).

3513 Exercises

- 3514 1. Prove that n -gram language models give valid probabilities if the n -gram probabili-
 3515 ties are valid. Specifically, assume that,

$$\sum_{w_m}^{\mathcal{V}} p(w_m | w_{m-1}, w_{m-2}, \dots, w_{m-n+1}) = 1 \quad [6.43]$$

3516 for all contexts $(w_{m-1}, w_{m-2}, \dots, w_{m-n+1})$. Prove that $\sum_{\mathbf{w}} p_n(\mathbf{w}) = 1$ for all $\mathbf{w} \in \mathcal{V}^*$,
 3517 where p_n is the probability under an n -gram language model. Your proof should
 3518 proceed by induction. You should handle the start-of-string case $p(w_1 | \underbrace{\square, \dots, \square}_{n-1})$,

3519 but you need not handle the end-of-string token.

- 3520 2. First, show that RNN language models are valid using a similar proof technique to
 3521 the one in the previous problem.

3522 Next, let $p_r(\mathbf{w})$ indicate the probability of \mathbf{w} under RNN r . An ensemble of RNN
 3523 language models computes the probability,

$$p(\mathbf{w}) = \frac{1}{R} \sum_{r=1}^R p_r(\mathbf{w}). \quad [6.44]$$

3524 Does an ensemble of RNN language models compute a valid probability?

- 3525 3. Consider a unigram language model over a vocabulary of size V . Suppose that a
 3526 word appears m times in a corpus with M tokens in total. With Lidstone smoothing
 3527 of α , for what values of m is the smoothed probability greater than the unsmoothed
 3528 probability?
- 3529 4. Consider a simple language in which each token is drawn from the vocabulary \mathcal{V}
 3530 with probability $\frac{1}{V}$, independent of all other tokens.

Given a corpus of size M , what is the expectation of the fraction of all possible bigrams that have zero count? You may assume V is large enough that $\frac{1}{V} \approx \frac{1}{V-1}$.

Continuing the previous problem, determine the value of M such that the fraction of bigrams with zero count is at most $\epsilon \in (0, 1)$. As a hint, you may use the approximation $\ln(1 + \alpha) \approx \alpha$ for $\alpha \approx 0$.

In real languages, word probabilities are neither uniform nor independent. Assume that word probabilities are independent but not uniform, so that in general $p(w) \neq \frac{1}{V}$. Prove that the expected fraction of unseen bigrams will be higher than in the IID case.

Consider a recurrent neural network with a single hidden unit and a sigmoid activation, $h_m = \sigma(\theta h_{m-1} + x_m)$. Prove that if $|\theta| < 1$, then the gradient $\frac{\partial h_m}{\partial h_{m-k}}$ goes to zero as $k \rightarrow \infty$.⁷

Zipf's law states that if the word types in a corpus are sorted by frequency, then the frequency of the word at rank r is proportional to r^{-s} , where s is a free parameter, usually around 1. (Another way to view Zipf's law is that a plot of log frequency against log rank will be linear.) Solve for s using the counts of the first and second most frequent words, c_1 and c_2 .

Download the wikitext-2 dataset.⁸ Read in the training data and compute word counts. Estimate the Zipf's law coefficient by,

$$\hat{s} = \exp \left(\frac{(\log r) \cdot (\log c)}{\|\log r\|_2^2} \right), \quad [6.45]$$

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

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

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

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

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

3559 **Chapter 7**

3560 **Sequence labeling**

3561 The goal of sequence labeling is to assign tags to words, or more generally, to assign
3562 discrete labels to discrete elements in a sequence. There are many applications of se-
3563 quence labeling in natural language processing, and chapter 8 presents an overview. For
3564 now, we'll focus on the classic problem of **part-of-speech tagging**, which requires tagging
3565 each word by its grammatical category. Coarse-grained grammatical categories include
3566 **NOUNs**, which describe things, properties, or ideas, and **VERBs**, which describe actions
3567 and events. Consider a simple input:

3568 (7.1) They can fish.

3569 A dictionary of coarse-grained part-of-speech tags might include **NOUN** as the only valid
3570 tag for *they*, but both **NOUN** and **VERB** as potential tags for *can* and *fish*. An accurate se-
3571 quence labeling algorithm should select the verb tag for both *can* and *fish* in (7.1), but it
3572 should select noun for the same two words in the phrase *can of fish*.

3573 **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:

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

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

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

3574 Here the feature function takes three arguments as input: the sentence to be tagged (e.g.,
3575 *they can fish*), the proposed tag (e.g., N or V), and the index of the token to which this tag

3576 is applied. This simple feature function then returns a single feature: a tuple including
 3577 the word to be tagged and the tag that has been proposed. If the vocabulary size is V
 3578 and the number of tags is K , then there are $V \times K$ features. Each of these features must
 3579 be assigned a weight. These weights can be learned from a labeled dataset using a clas-
 3580 sification algorithm such as perceptron, but this isn't necessary in this case: it would be
 3581 equivalent to define the classification weights directly, with $\theta_{w,y} = 1$ for the tag y most
 3582 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 han-
 dle 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 ad-
 ditional features:

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

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

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

3583 These features contain enough information that a tagger should be able to choose the
 3584 right tag for the word *fish*: words that come after *can* are likely to be verbs, so the feature
 3585 $(w_{m-1} = \text{can}, y_m = \mathbf{V})$ should have a large positive weight.

3586 However, even with this enhanced feature set, it may be difficult to tag some se-
 3587 quences correctly. One reason is that there are often relationships between the tags them-
 3588 selves. For example, in English it is relatively rare for a verb to follow another verb —
 3589 particularly if we differentiate MODAL verbs like *can* and *should* from more typical verbs,
 3590 like *give*, *transcend*, and *befuddle*. We would like to incorporate preferences against tag se-
 3591 quences like VERB-VERB, and in favor of tag sequences like NOUN-VERB. The need for
 3592 such preferences is best illustrated by a **garden path sentence**:

3593 (7.2) The old man the boat.

3594 Grammatically, the word *the* is a DETERMINER. When you read the sentence, what
 3595 part of speech did you first assign to *old*? Typically, this word is an ADJECTIVE — abbrevi-
 3596 ated as J — which is a class of words that modify nouns. Similarly, *man* is usually a noun.
 3597 The resulting sequence of tags is D J N D N. But this is a mistaken “garden path” inter-
 3598 pretation, which ends up leading nowhere. It is unlikely that a determiner would directly

follow a noun,¹ and it is particularly unlikely that the entire sentence would lack a verb. The only possible verb in (7.2) is the word *man*, which can refer to the act of maintaining and piloting something — often boats. But if *man* is tagged as a verb, then *old* is seated between a determiner and a verb, and must be a noun. And indeed, adjectives often have a second interpretation as nouns when used in this way (e.g., *the young*, *the restless*). This reasoning, in which the labeling decisions are intertwined, cannot be applied in a setting where each tag is produced by an independent classification decision.

7.2 Sequence labeling as structure prediction

As an alternative, think of the entire sequence of tags as a label itself. For a given sequence of words $\mathbf{w} = (w_1, w_2, \dots, w_M)$, there is a set of possible taggings $\mathcal{Y}(\mathbf{w}) = \mathcal{Y}^M$, where $\mathcal{Y} = \{\text{N, V, D, ...}\}$ refers to the set of individual tags, and \mathcal{Y}^M refers to the set of tag sequences of length M . We can then treat the sequence labeling problem as a classification problem in the label space $\mathcal{Y}(\mathbf{w})$,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}), \quad [7.7]$$

where $\mathbf{y} = (y_1, y_2, \dots, y_M)$ is a sequence of M tags, and Ψ is a scoring function on pairs of sequences, $V^M \times \mathcal{Y}^M \rightarrow \mathbb{R}$. Such a function can include features that capture the relationships between tagging decisions, such as the preference that determiners not follow nouns, or that all sentences have verbs.

Given that the label space is exponentially large in the length of the sequence M , can it ever be practical to perform tagging in this way? The problem of making a series of interconnected labeling decisions is known as **inference**. Because natural language is full of interrelated grammatical structures, inference is a crucial aspect of natural language processing. In English, it is not unusual to have sentences of length $M = 20$; part-of-speech tag sets vary in size from 10 to several hundred. Taking the low end of this range, we have $|\mathcal{Y}(\mathbf{w}_{1:M})| \approx 10^{20}$, one hundred billion billion possible tag sequences. Enumerating and scoring each of these sequences would require an amount of work that is exponential in the sequence length, so inference is intractable.

However, the situation changes when we restrict the scoring function. Suppose we choose a function that decomposes into a sum of local parts,

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.8]$$

where each $\psi(\cdot)$ scores a local part of the tag sequence. Note that the sum goes up to $M+1$, so that we can include a score for a special end-of-sequence tag, $\psi(\mathbf{w}_{1:M}, \diamond, y_M, M+1)$. We also define a special tag to begin the sequence, $y_0 \triangleq \diamond$.

¹The main exception occurs with ditransitive verbs, such as *They gave the winner a trophy*.

3628 In a linear model, local scoring function can be defined as a dot product of weights
 3629 and features,

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

3630 The feature vector \mathbf{f} can consider the entire input \mathbf{w} , and can look at pairs of adjacent
 3631 tags. This is a step up from per-token classification: the weights can assign low scores
 3632 to infelicitous tag pairs, such as noun-determiner, and high scores for frequent tag pairs,
 3633 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**):

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

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

3634 There are seven active features for this example: one for each word-tag pair, and one
 3635 for each tag-tag pair, including a final tag $y_{M+1} = \blacklozenge$. These features capture the two main
 3636 sources of information for part-of-speech tagging in English: which tags are appropriate
 3637 for each word, and which tags tend to follow each other in sequence. Given appropriate
 3638 weights for these features, taggers can achieve high accuracy, even for difficult cases like
 3639 *the old man the boat*. We will now discuss how this restricted scoring function enables
 3640 efficient inference, through the **Viterbi algorithm** (Viterbi, 1967).

3641 **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:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) \quad [7.13]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.14]$$

$$= \operatorname{argmax}_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.15]$$

3642 where the final line simplifies the notation with the shorthand,

$$s_m(y_m, y_{m-1}) \triangleq \psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m). \quad [7.16]$$

This inference problem can be solved efficiently using **dynamic programming**, an algorithmic technique for reusing work in recurrent computations. We begin by solving an auxiliary problem: rather than finding the best tag sequence, we compute the *score* of the best tag sequence,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{\mathbf{y}_{1:M}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \quad [7.17]$$

This score involves a maximization over all tag sequences of length M , written $\max_{\mathbf{y}_{1:M}}$. This maximization can be broken into two pieces,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \max_{y_M} \max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \quad [7.18]$$

Within the sum, only the final term $s_{M+1}(\blacklozenge, y_M)$ depends on y_M , so we can pull this term out of the second maximization,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}, \mathbf{y}_{1:M}) = \left(\max_{y_M} s_{M+1}(\blacklozenge, y_M) \right) + \left(\max_{\mathbf{y}_{1:M-1}} \sum_{m=1}^M s_m(y_m, y_{m-1}) \right). \quad [7.19]$$

The second term in Equation 7.19 has the same form as our original problem, with M replaced by $M-1$. This indicates that the problem can be reformulated as a recurrence. We do this by defining an auxiliary variable called the **Viterbi variable** $v_m(k)$, representing

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  $\mathbf{y}_{1:M}$ 
```

the score of the best sequence terminating in the tag k :

$$v_m(y_m) \triangleq \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.20]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \sum_{n=1}^{m-1} s_n(y_n, y_{n-1}) \quad [7.21]$$

$$= \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.22]$$

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 score for the first tag,

$$v_1(y_1) \triangleq s_1(y_1, \diamond). \quad [7.23]$$

The maximum overall score for the sequence is then the final Viterbi variable,

$$\max_{\mathbf{y}_{1:M}} \Psi(\mathbf{w}_{1:M}, \mathbf{y}_{1:M}) = v_{M+1}(\blacklozenge). \quad [7.24]$$

3643 Thus, the score of the best labeling for the sequence can be computed in a single forward
 3644 sweep: first compute all variables $v_1(\cdot)$ from Equation 7.23, and then compute all variables
 3645 $v_2(\cdot)$ from the recurrence in Equation 7.22, continuing until the final variable $v_{M+1}(\blacklozenge)$.

3646 The Viterbi variables can be arranged in a structure known as a **trellis**, shown in Fig-
 3647 ure 7.1. Each column indexes a token m in the sequence, and each row indexes a tag in
 3648 \mathcal{Y} ; every $v_{m-1}(k)$ is connected to every $v_m(k')$, indicating that $v_m(k')$ is computed from
 3649 $v_{m-1}(k)$. Special nodes are set aside for the start and end states.

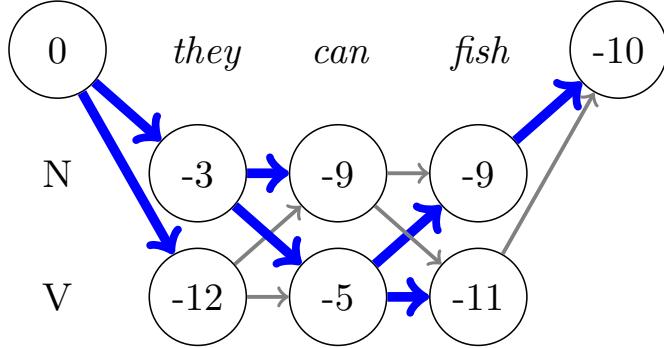


Figure 7.1: The trellis representation of the Viterbi variables, for the example *they can fish*, using the weights shown in Table 7.1.

3650 The original goal was to find the best scoring sequence, not simply to compute its
 3651 score. But by solving the auxiliary problem, we are almost there. Recall that each $v_m(k)$
 3652 represents the score of the best tag sequence ending in that tag k in position m . To compute
 3653 this, we maximize over possible values of y_{m-1} . By keeping track of the “argmax” tag that
 3654 maximizes this choice at each step, we can walk backwards from the final tag, and recover
 3655 the optimal tag sequence. This is indicated in Figure 7.1 by the thick lines, which we trace
 3656 back from the final position. These backward pointers are written $b_m(k)$, indicating the
 3657 optimal tag y_{m-1} on the path to $Y_m = k$.

3658 The complete Viterbi algorithm is shown in Algorithm 11. When computing the initial
 3659 Viterbi variables $v_1(\cdot)$, the special tag \diamond indicates the start of the sequence. When comput-
 3660 ing the final tag Y_M , another special tag, \blacklozenge indicates the end of the sequence. These special
 3661 tags enable the use of transition features for the tags that begin and end the sequence: for
 3662 example, conjunctions are unlikely to end sentences in English, so we would like a low
 3663 score for $s_{M+1}(\blacklozenge, CC)$; nouns are relatively likely to appear at the beginning of sentences,
 3664 so we would like a high score for $s_1(N, \diamond)$, assuming the noun tag is compatible with the
 3665 first word token w_1 .

3666 **Complexity** If there are K tags and M positions in the sequence, then there are $M \times K$
 3667 Viterbi variables to compute. Computing each variable requires finding a maximum over
 3668 K possible predecessor tags. The total time complexity of populating the trellis is there-
 3669 fore $\mathcal{O}(MK^2)$, with an additional factor for the number of active features at each position.
 3670 After completing the trellis, we simply trace the backwards pointers to the beginning of
 3671 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 $-\infty$.3672 **7.3.1 Example**

3673 Consider the minimal tagset $\{N, V\}$, corresponding to nouns and verbs. Even in this
 3674 tagset, there is considerable ambiguity: for example, the words *can* and *fish* can each take
 3675 both tags. Of the $2 \times 2 \times 2 = 8$ possible taggings for the sentence *they can fish*, four are
 3676 possible given these possible tags, and two are grammatical.²

3677 The values in the trellis in Figure 7.1 are computed from the feature weights defined in
 3678 Table 7.1. We begin with $v_1(N)$, which has only one possible predecessor, the start tag \diamond .
 3679 This score is therefore equal to $s_1(N, \diamond) = -2 - 1 = -3$, which is the sum of the scores for
 3680 the emission and transition features respectively; the backpointer is $b_1(N) = \diamond$. The score
 3681 for $v_1(V)$ is computed in the same way: $s_1(V, \diamond) = -10 - 2 = -12$, and again $b_1(V) = \diamond$.
 3682 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,

$$v_4(\diamond) = \max(v_3(N) + s_4(\diamond, N), v_3(V) + s_4(\diamond, V)) \quad [7.28]$$

$$= \max(-9 + 0 - 1, -11 + 0 - 1) \quad [7.29]$$

$$= -10, \quad [7.30]$$

3683 so $b_4(\diamond) = N$. As there is no emission w_4 , the emission features have scores of zero.

²The tagging *they/N can/V fish/N* corresponds to the scenario of putting fish into cans, or perhaps of firing them.

3684 To compute the optimal tag sequence, we walk backwards from here, next checking
 3685 $b_3(N) = V$, and then $b_2(V) = N$, and finally $b_1(N) = \diamond$. This yields $y = (N, V, N)$, which
 3686 corresponds to the linguistic interpretation of the fishes being put into cans.

3687 **7.3.2 Higher-order features**

3688 The Viterbi algorithm was made possible by a restriction of the scoring function to local
 3689 parts that consider only pairs of adjacent tags. We can think of this as a bigram language
 3690 model over tags. A natural question is how to generalize Viterbi to tag trigrams, which
 3691 would involve the following decomposition:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+2} f(\mathbf{w}, y_m, y_{m-1}, y_{m-2}, m), \quad [7.31]$$

3692 where $y_{-1} = \diamond$ and $y_{M+2} = \blacklozenge$.

3693 One solution is to create a new tagset $\mathcal{Y}^{(2)}$ from the Cartesian product of the original
 3694 tagset with itself, $\mathcal{Y}^{(2)} = \mathcal{Y} \times \mathcal{Y}$. The tags in this product space are ordered pairs, rep-
 3695 resenting adjacent tags at the token level: for example, the tag (N, V) would represent a
 3696 noun followed by a verb. Transitions between such tags must be consistent: we can have a
 3697 transition from (N, V) to (V, N) (corresponding to the tag sequence $N V N$), but not from
 3698 (N, V) to (N, N) , which would not correspond to any coherent tag sequence. This con-
 3699 straint can be enforced in feature weights, with $\theta_{((a,b),(c,d))} = -\infty$ if $b \neq c$. The remaining
 3700 feature weights can encode preferences for and against various tag trigrams.

3701 In the Cartesian product tag space, there are K^2 tags, suggesting that the time com-
 3702 plexity will increase to $\mathcal{O}(MK^4)$. However, it is unnecessary to max over predecessor tag
 3703 bigrams that are incompatible with the current tag bigram. By exploiting this constraint,
 3704 it is possible to limit the time complexity to $\mathcal{O}(MK^3)$. The space complexity grows to
 3705 $\mathcal{O}(MK^2)$, since the trellis must store all possible predecessors of each tag. In general, the
 3706 time and space complexity of higher-order Viterbi grows exponentially with the order of
 3707 the tag n -grams that are considered in the feature decomposition.

3708 **7.4 Hidden Markov Models**

3709 The Viterbi sequence labeling algorithm is built on the scores $s_m(y, y')$. We will now
 3710 discuss how these scores can be estimated probabilistically. Recall from § 2.2 that the
 3711 probabilistic Naïve Bayes classifier selects the label y to maximize $p(y | \mathbf{x}) \propto p(y, \mathbf{x})$. In
 3712 probabilistic sequence labeling, our goal is similar: select the tag sequence that maximizes
 3713 $p(y | \mathbf{w}) \propto p(y, \mathbf{w})$. The locality restriction in Equation 7.8 can be viewed as a conditional
 3714 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

```

3715 Naïve Bayes was introduced as a **generative model** — a probabilistic story that ex-
 3716 plains the observed data as well as the hidden label. A similar story can be constructed
 3717 for probabilistic sequence labeling: first, the tags are drawn from a prior distribution; next,
 3718 the tokens are drawn from a conditional likelihood. However, for inference to be tractable,
 3719 additional independence assumptions are required. First, the probability of each token
 3720 depends only on its tag, and not on any other element in the sequence:

$$p(w | y) = \prod_{m=1}^M p(w_m | y_m). \quad [7.32]$$

3721 Second, each tag y_m depends only on its predecessor,

$$p(y) = \prod_{m=1}^M p(y_m | y_{m-1}), \quad [7.33]$$

3722 where $y_0 = \diamond$ in all cases. Due to this **Markov assumption**, probabilistic sequence labeling
 3723 models are known as **hidden Markov models** (HMMs).

3724 The generative process for the hidden Markov model is shown in Algorithm 12. Given
 3725 the parameters λ and ϕ , we can compute $p(w, y)$ for any token sequence w and tag se-
 3726 quence y . The HMM is often represented as a **graphical model** (Wainwright and Jordan,
 3727 2008), as shown in Figure 7.2. This representation makes the independence assumptions
 3728 explicit: if a variable v_1 is probabilistically conditioned on another variable v_2 , then there
 3729 is an arrow $v_2 \rightarrow v_1$ in the diagram. If there are no arrows between v_1 and v_2 , they
 3730 are **conditionally independent**, given each variable's **Markov blanket**. In the hidden
 3731 Markov model, the Markov blanket for each tag y_m includes the “parent” y_{m-1} , and the
 3732 “children” y_{m+1} and w_m .³

3733 It is important to reflect on the implications of the HMM independence assumptions.
 3734 A non-adjacent pair of tags y_m and y_n are conditionally independent; if $m < n$ and we
 3735 are given y_{n-1} , then y_m offers no additional information about y_n . However, if we are
 3736 not given any information about the tags in a sequence, then all tags are probabilistically
 3737 coupled.

³In general graphical models, a variable's Markov blanket includes its parents, children, and its children's other parents (Murphy, 2012).

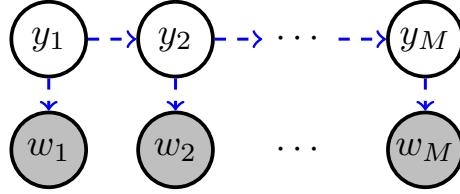


Figure 7.2: Graphical representation of the hidden Markov model. Arrows indicate probabilistic dependencies.

3738 7.4.1 Estimation

3739 The hidden Markov model has two groups of parameters:

3740 **Emission probabilities.** The probability $p_e(w_m | y_m; \phi)$ is the emission probability, since
3741 the words are treated as probabilistically “emitted”, conditioned on the tags.

3742 **Transition probabilities.** The probability $p_t(y_m | y_{m-1}; \lambda)$ is the transition probability,
3743 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,

$$\begin{aligned}\phi_{k,i} &\triangleq \Pr(W_m = i | Y_m = k) = \frac{\text{count}(W_m = i, Y_m = k)}{\text{count}(Y_m = k)} \\ \lambda_{k,k'} &\triangleq \Pr(Y_m = k' | Y_{m-1} = k) = \frac{\text{count}(Y_m = k', Y_{m-1} = k)}{\text{count}(Y_{m-1} = k)}.\end{aligned}$$

3744 Smoothing is more important for the emission probability than the transition probability,
3745 because the vocabulary is much larger than the number of tags.

3746 7.4.2 Inference

3747 The goal of inference in the hidden Markov model is to find the highest probability tag
3748 sequence,

$$\hat{y} = \underset{y}{\operatorname{argmax}} p(y | w). \quad [7.34]$$

3749 As in Naïve Bayes, it is equivalent to find the tag sequence with the highest *log*-probability,
3750 since the logarithm is a monotonically increasing function. It is furthermore equivalent
3751 to maximize the joint probability $p(y, w) = p(y | w) \times p(w) \propto p(y | w)$, which is pro-
3752 portional to the conditional probability. Putting these observations together, the inference

3753 problem can be reformulated as,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y}, \mathbf{w}). \quad [7.35]$$

We can now apply the HMM independence assumptions:

$$\log p(\mathbf{y}, \mathbf{w}) = \log p(\mathbf{y}) + \log p(\mathbf{w} \mid \mathbf{y}) \quad [7.36]$$

$$= \sum_{m=1}^{M+1} \log p_Y(y_m \mid y_{m-1}) + \log p_{W|Y}(w_m \mid y_m) \quad [7.37]$$

$$= \sum_{m=1}^{M+1} \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m} \quad [7.38]$$

$$= \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}), \quad [7.39]$$

where,

$$s_m(y_m, y_{m-1}) \triangleq \log \lambda_{y_m, y_{m-1}} + \log \phi_{y_m, w_m}, \quad [7.40]$$

3754 and,

$$\phi_{\diamond, w} = \begin{cases} 1, & w = \blacksquare \\ 0, & \text{otherwise,} \end{cases} \quad [7.41]$$

3755 which ensures that the stop tag \diamond can only be applied to the final token \blacksquare .

This derivation shows that HMM inference can be viewed as an application of the Viterbi decoding algorithm, given an appropriately defined scoring function. The local score $s_m(y_m, y_{m-1})$ can be interpreted probabilistically,

$$s_m(y_m, y_{m-1}) = \log p_y(y_m \mid y_{m-1}) + \log p_{w|y}(w_m \mid y_m) \quad [7.42]$$

$$= \log p(y_m, w_m \mid y_{m-1}). \quad [7.43]$$

Now recall the definition of the Viterbi variables,

$$v_m(y_m) = \max_{y_{m-1}} s_m(y_m, y_{m-1}) + v_{m-1}(y_{m-1}) \quad [7.44]$$

$$= \max_{y_{m-1}} \log p(y_m, w_m \mid y_{m-1}) + v_{m-1}(y_{m-1}). \quad [7.45]$$

By setting $v_{m-1}(y_{m-1}) = \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1})$, we obtain the recurrence,

$$v_m(y_m) = \max_{y_{m-1}} \log p(y_m, w_m \mid y_{m-1}) + \max_{\mathbf{y}_{1:m-2}} \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.46]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(y_m, w_m \mid y_{m-1}) + \log p(\mathbf{y}_{1:m-1}, \mathbf{w}_{1:m-1}) \quad [7.47]$$

$$= \max_{\mathbf{y}_{1:m-1}} \log p(\mathbf{y}_{1:m}, \mathbf{w}_{1:m}). \quad [7.48]$$

In words, the Viterbi variable $v_m(y_m)$ is the log probability of the best tag sequence ending in y_m , joint with the word sequence $w_{1:m}$. The log probability of the best complete tag sequence is therefore,

$$\max_{\mathbf{y}_{1:M}} \log p(\mathbf{y}_{1:M+1}, \mathbf{w}_{1:M+1}) = v_{M+1}(\spadesuit) \quad [7.49]$$

***Viterbi as an example of the max-product algorithm** The Viterbi algorithm can also be implemented using probabilities, rather than log-probabilities. In this case, each $v_m(y_m)$ is equal to,

$$v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} p(\mathbf{y}_{1:m-1}, y_m, \mathbf{w}_{1:m}) \quad [7.50]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times \max_{\mathbf{y}_{1:m-2}} p(\mathbf{y}_{1:m-2}, y_{m-1}, \mathbf{w}_{1:m-1}) \quad [7.51]$$

$$= \max_{y_{m-1}} p(y_m, w_m | y_{m-1}) \times v_{m-1}(y_{m-1}) \quad [7.52]$$

$$= p_{w|y}(w_m | y_m) \times \max_{y_{m-1}} p_y(y_m | y_{m-1}) \times v_{m-1}(y_{m-1}). \quad [7.53]$$

3756 Each Viterbi variable is computed by *maximizing* over a set of *products*. Thus, the Viterbi
 3757 algorithm is a special case of the **max-product algorithm** for inference in graphical mod-
 3758 els (Wainwright and Jordan, 2008). However, the product of probabilities tends towards
 3759 zero over long sequences, so the log-probability version of Viterbi is recommended in
 3760 practical implementations.

3761 7.5 Discriminative sequence labeling with features

3762 Today, hidden Markov models are rarely used for supervised sequence labeling. This is
 3763 because HMMs are limited to only two phenomena:

- 3764 • word-tag compatibility, via the emission probability $p_{W|Y}(w_m | y_m)$;
- 3765 • local context, via the transition probability $p_Y(y_m | y_{m-1})$.

3766 The Viterbi algorithm permits the inclusion of richer information in the local scoring func-
 3767 tion $\psi(\mathbf{w}_{1:M}, y_m, y_{m-1}, m)$, which can be defined as a weighted sum of arbitrary local *fea-*
 3768 *tures*,

$$\psi(\mathbf{w}, y_m, y_{m-1}, m) = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m), \quad [7.54]$$

3769 where \mathbf{f} is a locally-defined feature function, and $\boldsymbol{\theta}$ is a vector of weights.

The local decomposition of the scoring function Ψ is reflected in a corresponding decomposition of the feature function:

$$\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.55]$$

$$= \theta \cdot \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.56]$$

$$= \theta \cdot \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}, y_m, y_{m-1}, m) \quad [7.57]$$

$$= \theta \cdot \mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}_{1:M}), \quad [7.58]$$

3770 where $\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y})$ is a global feature vector, which is a sum of local feature vectors,

$$\mathbf{f}^{(\text{global})}(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \mathbf{f}(\mathbf{w}_{1:M}, y_m, y_{m-1}, m), \quad [7.59]$$

3771 with $y_{M+1} = \diamond$ and $y_0 = \diamond$ by construction.

3772 Let's now consider what additional information these features might encode.

3773 **Word affix features.** Consider the problem of part-of-speech tagging on the first four
3774 lines of the poem *Jabberwocky* (Carroll, 1917):

3775 (7.3) 'Twas brillig, and the slithy toves
3776 Did gyre and gimble in the wabe:
3777 All mimsy were the borogoves,
3778 And the mome raths outgrabe.

3779 Many of these words were made up by the author of the poem, so a corpus would offer
3780 no information about their probabilities of being associated with any particular part of
3781 speech. Yet it is not so hard to see what their grammatical roles might be in this passage.
3782 Context helps: for example, the word *slithy* follows the determiner *the*, so it is probably a
3783 noun or adjective. Which do you think is more likely? The suffix *-thy* is found in a number
3784 of adjectives, like *frothy*, *healthy*, *pithy*, *worthy*. It is also found in a handful of nouns — e.g.,
3785 *apathy*, *sympathy* — but nearly all of these have the longer coda *-pathy*, unlike *slithy*. So the
3786 suffix gives some evidence that *slithy* is an adjective, and indeed it is: later in the text we
3787 find that it is a combination of the adjectives *lithe* and *slimy*.⁴

⁴Morphology is the study of how words are formed from smaller linguistic units. Chapter 9 touches on computational approaches to morphological analysis. See Bender (2013) for an overview of the underlying linguistic principles, and Haspelmath and Sims (2013) or Lieber (2015) for a full treatment.

3788 **Fine-grained context.** The hidden Markov model captures contextual information in the
 3789 form of part-of-speech tag bigrams. But sometimes, the necessary contextual information
 3790 is more specific. Consider the noun phrases *this fish* and *these fish*. Many part-of-speech
 3791 tagsets distinguish between singular and plural nouns, but do not distinguish between
 3792 singular and plural determiners; for example, the well known **Penn Treebank** tagset fol-
 3793 lows these conventions. A hidden Markov model would be unable to correctly label *fish* as
 3794 singular or plural in both of these cases, because it only has access to two features: the pre-
 3795 ceding tag (determiner in both cases) and the word (*fish* in both cases). The classification-
 3796 based tagger discussed in § 7.1 had the ability to use preceding and succeeding words as
 3797 features, and it can also be incorporated into a Viterbi-based sequence labeler as a local
 3798 feature.

Example. Consider the tagging D J N (determiner, adjective, noun) for the sequence *the slithy toves*, so that

$$\begin{aligned} \mathbf{w} &= \text{the slithy toves} \\ \mathbf{y} &= \text{D J N}. \end{aligned}$$

Let's create the feature vector for this example, assuming that we have word-tag features (indicated by W), tag-tag features (indicated by T), and suffix features (indicated by M). You can assume that you have access to a method for extracting the suffix *-thy* from *slithy*, *-es* from *toves*, and \emptyset from *the*, indicating that this word has no suffix.⁵ The resulting feature vector is,

$$\begin{aligned} \mathbf{f}(\text{the slithy toves}, \text{D J N}) &= \mathbf{f}(\text{the slithy toves}, \text{D}, \diamond, 1) \\ &\quad + \mathbf{f}(\text{the slithy toves}, \text{J}, \text{D}, 2) \\ &\quad + \mathbf{f}(\text{the slithy toves}, \text{N}, \text{J}, 3) \\ &\quad + \mathbf{f}(\text{the slithy toves}, \blacklozenge, \text{N}, 4) \\ &= \{(T : \diamond, \text{D}), (W : \text{the}, \text{D}), (M : \emptyset, \text{D}), \\ &\quad (T : \text{D}, \text{J}), (W : \text{slithy}, \text{J}), (M : \text{-thy}, \text{J}), \\ &\quad (T : \text{J}, \text{N}), (W : \text{toves}, \text{N}), (M : \text{-es}, \text{N}) \\ &\quad (T : \text{N}, \blacklozenge)\}. \end{aligned}$$

3799 These examples show that local features can incorporate information that lies beyond
 3800 the scope of a hidden Markov model. Because the features are local, it is possible to apply
 3801 the Viterbi algorithm to identify the optimal sequence of tags. The remaining question

⁵Such a system is called a **morphological segmenter**. The task of morphological segmentation is briefly described in § 9.1.4; 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*.

3802 is how to estimate the weights on these features. § 2.3 presented three main types of
 3803 discriminative classifiers: perceptron, support vector machine, and logistic regression.
 3804 Each of these classifiers has a structured equivalent, enabling it to be trained from labeled
 3805 sequences rather than individual tokens.

3806 **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,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \theta \cdot f(\mathbf{w}, \mathbf{y}) \quad [7.62]$$

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + f(\mathbf{w}, \mathbf{y}) - f(\mathbf{w}, \hat{\mathbf{y}}). \quad [7.63]$$

3807 This learning algorithm is called **structured perceptron**, because it learns to predict the
 3808 structured output \mathbf{y} . The only difference is that instead of computing \hat{y} by enumerating
 3809 the entire set \mathcal{Y} , the Viterbi algorithm is used to efficiently search the set of possible tag-
 3810 gings, \mathcal{Y}^M . Structured perceptron can be applied to other structured outputs as long as
 3811 efficient inference is possible. As in perceptron classification, weight averaging is crucial
 3812 to get good performance (see § 2.3.2).

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:

$$\theta_{(fish,V)} \leftarrow \theta_{(fish,V)} + 1, \quad \theta_{(fish,N)} \leftarrow \theta_{(fish,N)} - 1 \quad [7.64]$$

$$\theta_{(V,V)} \leftarrow \theta_{(V,V)} + 1, \quad \theta_{(V,N)} \leftarrow \theta_{(V,N)} - 1 \quad [7.65]$$

$$\theta_{(V,\blacklozenge)} \leftarrow \theta_{(V,\blacklozenge)} + 1, \quad \theta_{(N,\blacklozenge)} \leftarrow \theta_{(N,\blacklozenge)} - 1. \quad [7.66]$$

3813 **7.5.2 Structured support vector machines**

3814 Large-margin classifiers such as the support vector machine improve on the perceptron by
 3815 pushing the classification boundary away from the training instances. The same idea can

3816 be applied to sequence labeling. A support vector machine in which the output is a struc-
 3817 tured object, such as a sequence, is called a **structured support vector machine** (Tsochan-
 3818 taridis et al., 2004).⁶

3819 In classification, we formalized the large-margin constraint as,

$$\forall \mathbf{y} \neq \mathbf{y}^{(i)}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, \mathbf{y}) \geq 1, \quad [7.67]$$

3820 requiring a margin of at least 1 between the scores for all labels \mathbf{y} that are not equal to the
 3821 correct label $\mathbf{y}^{(i)}$. The weights $\boldsymbol{\theta}$ are then learned by constrained optimization (see § 2.4.2).

3822 This idea can be applied to sequence labeling by formulating an equivalent set of con-
 3823 straints for all possible labelings $\mathcal{Y}(\mathbf{w})$ for an input \mathbf{w} . However, there are two problems.
 3824 First, in sequence labeling, some predictions are more wrong than others: we may miss
 3825 only one tag out of fifty, or we may get all fifty wrong. We would like our learning algo-
 3826 rithm to be sensitive to this difference. Second, the number of constraints is equal to the
 3827 number of possible labelings, which is exponentially large in the length of the sequence.

3828 The first problem can be addressed by adjusting the constraint to require larger mar-
 3829 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
 3830 the true label is $\mathbf{y}^{(i)}$. We can then generalize the margin constraint,

$$\forall \mathbf{y}, \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) \geq c(\mathbf{y}^{(i)}, \mathbf{y}). \quad [7.68]$$

3831 This cost-augmented margin constraint specializes to the constraint in Equation 7.67 if we
 3832 choose the delta function $c(\mathbf{y}^{(i)}, \mathbf{y}) = \delta((\mathbf{y}^{(i)} \neq \mathbf{y}))$. A more expressive cost function is
 3833 the **Hamming cost**,

$$c(\mathbf{y}^{(i)}, \mathbf{y}) = \sum_{m=1}^M \delta(y_m^{(i)} \neq y_m), \quad [7.69]$$

3834 which computes the number of errors in \mathbf{y} . By incorporating the cost function as the
 3835 margin constraint, we require that the true labeling be separated from the alternatives by
 3836 a margin that is proportional to the number of incorrect tags in each alternative labeling.

The second problem is that the number of constraints is exponential in the length
 of the sequence. This can be addressed by focusing on the prediction $\hat{\mathbf{y}}$ that *maximally*
 violates the margin constraint. This prediction can be identified by solving the following
cost-augmented decoding problem:

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + c(\mathbf{y}^{(i)}, \mathbf{y}) \quad [7.70]$$

$$= \operatorname{argmax}_{\mathbf{y} \neq \mathbf{y}^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y}), \quad [7.71]$$

⁶This model is also known as a **max-margin Markov network** (Taskar et al., 2003), emphasizing that the scoring function is constructed from a sum of components, which are Markov independent.

3837 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,

$$\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})) \geq 0. \quad [7.72]$$

3838 If the score for $\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})$ is greater than the cost-augmented score for all alternatives,
 3839 then the constraint will be met. The name “cost-augmented decoding” is due to the fact
 3840 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
 3841 an additional term for the cost. Essentially, we want to train against predictions that are
 3842 strong and wrong: they should score highly according to the model, yet incur a large loss
 3843 with respect to the ground truth. Training adjusts the weights to reduce the score of these
 3844 predictions.

3845 For cost-augmented decoding to be tractable, the cost function must decompose into
 3846 local parts, just as the feature function $f(\cdot)$ does. The Hamming cost, defined above,
 3847 obeys this property. To perform cost-augmented decoding using the Hamming cost, we
 3848 need only to add features $f_m(y_m) = \delta(y_m \neq y_m^{(i)})$, and assign a constant weight of 1 to
 3849 these features. Decoding can then be performed using the Viterbi algorithm.⁷

As with large-margin classifiers, it is possible to formulate the learning problem in an unconstrained form, by combining a regularization term on the weights and a Lagrangian for the constraints:

$$\min_{\theta} \frac{1}{2} \|\theta\|_2^2 - C \left(\sum_i \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) - \max_{\mathbf{y} \in \mathcal{Y}(\mathbf{w}^{(i)})} [\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}) + c(\mathbf{y}^{(i)}, \mathbf{y})] \right), \quad [7.73]$$

3850 In this formulation, C is a parameter that controls the tradeoff between the regularization
 3851 term and the margin constraints. A number of optimization algorithms have been
 3852 proposed for structured support vector machines, some of which are discussed in § 2.4.2.
 3853 An empirical comparison by Kummerfeld et al. (2015) shows that stochastic subgradient
 3854 descent — which is essentially a cost-augmented version of the structured perceptron —
 3855 is highly competitive.

3856 7.5.3 Conditional random fields

3857 The **conditional random field** (CRF; Lafferty et al., 2001) is a conditional probabilistic
 3858 model for sequence labeling; just as structured perceptron is built on the perceptron clas-
 3859 sifier, conditional random fields are built on the logistic regression classifier.⁸ The basic

⁷Are there cost functions that do not decompose into local parts? Suppose we want to assign a constant loss c to any prediction $\hat{\mathbf{y}}$ in which k or more predicted tags are incorrect, and zero loss otherwise. This loss function is combinatorial over the predictions, and thus we cannot decompose it into parts.

⁸The name “conditional random field” is derived from **Markov random fields**, a general class of models in which the probability of a configuration of variables is proportional to a product of scores across pairs (or

3860 probability model is,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp(\Psi(\mathbf{w}, \mathbf{y}))}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp(\Psi(\mathbf{w}, \mathbf{y}'))}. \quad [7.74]$$

3861 This is almost identical to logistic regression (§ 2.5), but because the label space is now
 3862 sequences of tags, we require efficient algorithms for both **decoding** (searching for the
 3863 best tag sequence given a sequence of words \mathbf{w} and a model θ) and for **normalization**
 3864 (summing over all tag sequences). These algorithms will be based on the usual locality
 3865 assumption on the scoring function, $\Psi(\mathbf{w}, \mathbf{y}) = \sum_{m=1}^{M+1} \psi(\mathbf{w}, y_m, y_{m-1}, m)$.

3866 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 $p(\mathbf{y} \mid \mathbf{w})$,

$$\begin{aligned} \hat{\mathbf{y}} &= \operatorname{argmax}_{\mathbf{y}} \log p(\mathbf{y} \mid \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) - \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \Psi(\mathbf{y}', \mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{y}} \Psi(\mathbf{y}, \mathbf{w}) = \operatorname{argmax}_{\mathbf{y}} \sum_{m=1}^{M+1} s_m(y_m, y_{m-1}). \end{aligned}$$

3867 This is identical to the decoding problem for structured perceptron, so the same Viterbi
 3868 recurrence as defined in Equation 7.22 can be used.

3869 Learning in CRFs

As with logistic regression, the weights θ are learned by minimizing the regularized negative log-probability,

$$\ell = \frac{\lambda}{2} \|\theta\|^2 - \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \theta) \quad [7.75]$$

$$= \frac{\lambda}{2} \|\theta\|^2 - \sum_{i=1}^N \theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w}^{(i)})} \exp (\theta \cdot f(\mathbf{w}^{(i)}, \mathbf{y}')), \quad [7.76]$$

more generally, cliques) of variables in a **factor graph**. In sequence labeling, the pairs of variables include all adjacent tags (y_m, y_{m-1}). The probability is *conditioned* on the words \mathbf{w} , which are always observed, motivating the term “conditional” in the name.

3870 where λ controls the amount of regularization. The final term in Equation 7.76 is a sum
 3871 over all possible labelings. This term is the log of the denominator in Equation 7.74, some-
 3872 times known as the **partition function**.⁹ There are $|\mathcal{Y}|^M$ possible labelings of an input of
 3873 size M , so we must again exploit the decomposition of the scoring function to compute
 3874 this sum efficiently.

The sum $\sum_{\mathbf{y} \in \mathcal{Y}^{w(i)}} \exp \Psi(\mathbf{y}, \mathbf{w})$ can be computed efficiently using the **forward recurrence**, which is closely related to the Viterbi recurrence. We first define a set of **forward variables**, $\alpha_m(y_m)$, which is equal to the sum of the scores of all paths leading to tag y_m at position m :

$$\alpha_m(y_m) \triangleq \sum_{\mathbf{y}_{1:m-1}} \exp \sum_{n=1}^m s_n(y_n, y_{n-1}) \quad [7.77]$$

$$= \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}). \quad [7.78]$$

Note the similarity to the definition of the Viterbi variable, $v_m(y_m) = \max_{\mathbf{y}_{1:m-1}} \sum_{n=1}^m s_n(y_n, y_{n-1})$. In the hidden Markov model, the Viterbi recurrence had an alternative interpretation as the max-product algorithm (see Equation 7.53); analogously, the forward recurrence is known as the **sum-product algorithm**, because of the form of [7.78]. The forward variable can also be computed through a recurrence:

$$\alpha_m(y_m) = \sum_{\mathbf{y}_{1:m-1}} \prod_{n=1}^m \exp s_n(y_n, y_{n-1}) \quad [7.79]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \sum_{\mathbf{y}_{1:m-2}} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \quad [7.80]$$

$$= \sum_{y_{m-1}} (\exp s_m(y_m, y_{m-1})) \times \alpha_{m-1}(y_{m-1}). \quad [7.81]$$

Using the forward recurrence, it is possible to compute the denominator of the conditional probability,

$$\sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{w})} \Psi(\mathbf{w}, \mathbf{y}) = \sum_{\mathbf{y}_{1:M}} s_{M+1}(\blacklozenge, y_M) \prod_{m=1}^M s_m(y_m, y_{m-1}) \quad [7.82]$$

$$= \alpha_{M+1}(\blacklozenge). \quad [7.83]$$

⁹The terminology of “potentials” and “partition functions” comes from statistical mechanics (Bishop, 2006).

The conditional log-likelihood can be rewritten,

$$\ell = \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 - \sum_{i=1}^N \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}) + \log \alpha_{M+1}(\blacklozenge). \quad [7.84]$$

3875 Probabilistic programming environments, such as TORCH (Collobert et al., 2011) and
 3876 DYNET (Neubig et al., 2017), can compute the gradient of this objective using automatic
 3877 differentiation. The programmer need only implement the forward algorithm as a com-
 3878 putation graph.

As in logistic regression, the gradient of the likelihood with respect to the parameters is a difference between observed and expected feature counts:

$$\frac{d\ell}{d\theta_j} = \lambda\theta_j + \sum_{i=1}^N E[f_j(\mathbf{w}^{(i)}, \mathbf{y})] - f_j(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}), \quad [7.85]$$

3879 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-
 3880 quence $\mathbf{y}^{(i)}$. The expected feature counts are computed “under the hood” when automatic
 3881 differentiation is applied to Equation 7.84 (Eisner, 2016).

3882 Before the widespread use of automatic differentiation, it was common to compute
 3883 the feature expectations from marginal tag probabilities $p(y_m | \mathbf{w})$. These marginal prob-
 3884 abilities are sometimes useful on their own, and can be computed using the **forward-**
 3885 **backward algorithm**. This algorithm combines the forward recurrence with an equivalent
 3886 **backward recurrence**, which traverses the input from w_M back to w_1 .

3887 *Forward-backward algorithm

Marginal probabilities over tag bigrams can be written as,¹⁰

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\sum_{\mathbf{y}: Y_m=k, Y_{m-1}=k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1})}{\sum_{\mathbf{y}'} \prod_{n=1}^M \exp s_n(y'_n, y'_{n-1})}. \quad [7.86]$$

The numerator sums over all tag sequences that include the transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$. Because we are only interested in sequences that include the tag bigram, this sum can be decomposed into three parts: the *prefixes* $\mathbf{y}_{1:m-1}$, terminating in $Y_{m-1} = k'$; the

¹⁰Recall the notational convention of upper-case letters for random variables, e.g. Y_m , and lower case letters for specific values, e.g., y_m , so that $Y_m = k$ is interpreted as the event of random variable Y_m taking the value k .

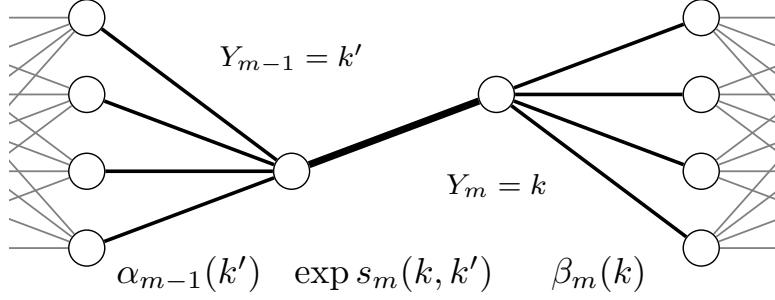


Figure 7.3: A schematic illustration of the computation of the marginal probability $\Pr(Y_{m-1} = k', Y_m = k)$, using the forward score $\alpha_{m-1}(k')$ and the backward score $\beta_m(k)$.

transition $(Y_{m-1} = k') \rightarrow (Y_m = k)$; and the *suffixes* $\mathbf{y}_{m:M}$, beginning with the tag $Y_m = k$:

$$\sum_{\mathbf{y}: Y_m = k, Y_{m-1} = k'} \prod_{n=1}^M \exp s_n(y_n, y_{n-1}) = \sum_{\mathbf{y}_{1:m-1}: Y_{m-1} = k'} \prod_{n=1}^{m-1} \exp s_n(y_n, y_{n-1}) \times \exp s_m(k, k') \times \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}). \quad [7.87]$$

The result is product of three terms: a score that sums over all the ways to get to the position $(Y_{m-1} = k')$, a score for the transition from k' to k , and a score that sums over all the ways of finishing the sequence from $(Y_m = k)$. The first term of Equation 7.87 is equal to the **forward variable**, $\alpha_{m-1}(k')$. The third term — the sum over ways to finish the sequence — can also be defined recursively, this time moving over the trellis from right to left, which is known as the **backward recurrence**:

$$\beta_m(k) \triangleq \sum_{\mathbf{y}_{m:M}: Y_m = k} \prod_{n=m}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.88]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \sum_{\mathbf{y}_{m+1:M}: Y_m = k'} \prod_{n=m+1}^{M+1} \exp s_n(y_n, y_{n-1}) \quad [7.89]$$

$$= \sum_{k' \in \mathcal{Y}} \exp s_{m+1}(k', k) \times \beta_{m+1}(k'). \quad [7.90]$$

3888 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,

$$p(y | \mathbf{w}_{1:m}) = \frac{\exp \psi_m(y)}{\sum_{y' \in \mathcal{Y}} \exp \psi_m(y')}. \quad [7.95]$$

3907 Using this transformation, it is possible to train the tagger from the negative log-likelihood
 3908 of the tags, as in a conditional random field. Alternatively, a hinge loss or margin loss
 3909 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:

$$\overleftarrow{\mathbf{h}}_m = g(\mathbf{x}_m, \overleftarrow{\mathbf{h}}_{m+1}), \quad m = 1, 2, \dots, M. \quad [7.96]$$

3910 The hidden states of the left-to-right RNN are denoted $\overrightarrow{\mathbf{h}}_m$. The left-to-right and right-to-
 3911 left vectors are concatenated, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. The scoring function in Equation 7.93 is
 3912 applied to this concatenated vector.

3913 Bidirectional RNN tagging has several attractive properties. Ideally, the representa-
 3914 tion \mathbf{h}_m summarizes the useful information from the surrounding context, so that it is not
 3915 necessary to design explicit features to capture this information. If the vector \mathbf{h}_m is an ad-
 3916 equate summary of this context, then it may not even be necessary to perform the tagging
 3917 jointly: in general, the gains offered by joint tagging of the entire sequence are diminished
 3918 as the individual tagging model becomes more powerful. Using backpropagation, the
 3919 word vectors \mathbf{x} can be trained “end-to-end”, so that they capture word properties that are
 3920 useful for the tagging task. Alternatively, if limited labeled data is available, we can use
 3921 word embeddings that are “pre-trained” from unlabeled data, using a language modeling
 3922 objective (as in § 6.3) or a related word embedding technique (see chapter 14). It is even
 3923 possible to combine both fine-tuned and pre-trained embeddings in a single model.

3924 **Neural structure prediction** The bidirectional recurrent neural network incorporates in-
 3925 formation from throughout the input, but each tagging decision is made independently.
 3926 In some sequence labeling applications, there are very strong dependencies between tags:
 3927 it may even be impossible for one tag to follow another. In such scenarios, the tagging
 3928 decision must be made jointly across the entire sequence.

3929 Neural sequence labeling can be combined with the Viterbi algorithm by defining the
 3930 local scores as:

$$s_m(y_m, y_{m-1}) = \beta_{y_m} \cdot \mathbf{h}_m + \eta_{y_{m-1}, y_m}, \quad [7.97]$$

3931 where \mathbf{h}_m is the RNN hidden state, β_{y_m} is a vector associated with tag y_m , and η_{y_{m-1}, y_m}
 3932 is a scalar parameter for the tag transition (y_{m-1}, y_m) . These local scores can then be
 3933 incorporated into the Viterbi algorithm for inference, and into the forward algorithm for
 3934 training. This model is shown in Figure 7.4. It can be trained from the conditional log-
 3935 likelihood objective defined in Equation 7.76, backpropagating to the tagging parameters

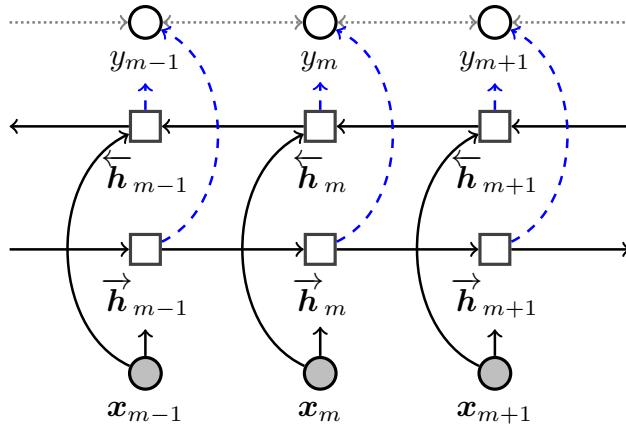


Figure 7.4: **Bidirectional LSTM** for sequence labeling. The solid lines indicate computation, the dashed lines indicate probabilistic dependency, and the dotted lines indicate the optional additional probabilistic dependencies between labels in the biLSTM-CRF.

3936 β and η , as well as the parameters of the RNN. This model is called the **LSTM-CRF**, due
 3937 to its combination of aspects of the long short-term memory and conditional random field
 3938 models (Huang et al., 2015).

3939 The LSTM-CRF is especially effective on the task of **named entity recognition** (Lample
 3940 et al., 2016), a sequence labeling task that is described in detail in § 8.3. This task has strong
 3941 dependencies between adjacent tags, so structure prediction is especially important.

3942 7.6.2 Character-level models

3943 As in language modeling, rare and unseen words are a challenge: if we encounter a word
 3944 that was not in the training data, then there is no obvious choice for the word embed-
 3945 ding x_m . One solution is to use a generic **unseen word** embedding for all such words.
 3946 However, in many cases, properties of unseen words can be guessed from their spellings.
 3947 For example, *whimsical* does not appear in the Universal Dependencies (UD) English Tree-
 3948 bank, yet the suffix *-al* makes it likely to be adjective; by the same logic, *unflinchingly* is
 3949 likely to be an adverb, and *barnacle* is likely to be a noun.

3950 In feature-based models, these morphological properties were handled by suffix fea-
 3951 tures; in a neural network, they can be incorporated by constructing the embeddings of
 3952 unseen words from their spellings or morphology. One way to do this is to incorporate
 3953 an additional layer of bidirectional RNNs, one for each word in the vocabulary (Ling
 3954 et al., 2015). For each such character-RNN, the inputs are the characters, and the output
 3955 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) with expectation-maximization (EM; § 5.1.2). In the M-step, the HMM parameters from expected counts:

$$\Pr(W = i \mid Y = k) = \phi_{k,i} = \frac{E[\text{count}(W = i, Y = k)]}{E[\text{count}(Y = k)]}$$

$$\Pr(Y_m = k \mid Y_{m-1} = k') = \lambda_{k',k} = \frac{E[\text{count}(Y_m = k, Y_{m-1} = k')]}{E[\text{count}(Y_{m-1} = k')]} \quad 3981$$

3982 The expected counts are computed in the E-step, using the forward and backward
 3983 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, these marginal probabilities can be computed from the forward-backward recurrence,

$$\Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}) = \frac{\alpha_{m-1}(k') \times \exp s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)}. \quad [7.101]$$

In a hidden Markov model, each element of the forward-backward computation has a special interpretation:

$$\alpha_{m-1}(k') = p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \quad [7.102]$$

$$\exp s_m(k, k') = p(Y_m = k, w_m | Y_{m-1} = k') \quad [7.103]$$

$$\beta_m(k) = p(\mathbf{w}_{m+1:M} | Y_m = k). \quad [7.104]$$

Applying the conditional independence assumptions of the hidden Markov model (defined in Algorithm 12), the product is equal to the joint probability of the tag bigram and the entire input,

$$\begin{aligned} \alpha_{m-1}(k') \times \exp s_m(k, k') \times \beta_m(k) &= p(Y_{m-1} = k', \mathbf{w}_{1:m-1}) \\ &\quad \times p(Y_m = k, w_m | Y_{m-1} = k') \\ &\quad \times p(\mathbf{w}_{m+1:M} | Y_m = k) \\ &= p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M}). \end{aligned} \quad [7.105]$$

Dividing by $\alpha_{M+1}(\blacklozenge) = p(\mathbf{w}_{1:M})$ gives the desired probability,

$$\frac{\alpha_{m-1}(k') \times s_m(k, k') \times \beta_m(k)}{\alpha_{M+1}(\blacklozenge)} = \frac{p(Y_{m-1} = k', Y_m = k, \mathbf{w}_{1:M})}{p(\mathbf{w}_{1:M})} \quad [7.106]$$

$$= \Pr(Y_{m-1} = k', Y_m = k | \mathbf{w}_{1:M}). \quad [7.107]$$

3984 The expected emission counts can be computed in a similar manner, using the product
 3985 $\alpha_m(k) \times \beta_m(k)$.

3986 **7.7.1 Linear dynamical systems**

3987 The forward-backward algorithm can be viewed as Bayesian state estimation in a discrete
 3988 state space. In a continuous state space, $y_m \in \mathbb{R}^K$, the equivalent algorithm is the **Kalman**
 3989 **smoother**. It also computes marginals $p(y_m | x_{1:M})$, using a similar two-step algorithm
 3990 of forward and backward passes. Instead of computing a trellis of values at each step, the
 3991 Kalman smoother computes a probability density function $q_{y_m}(y_m; \mu_m, \Sigma_m)$, character-
 3992 ized by a mean μ_m and a covariance Σ_m around the latent state. Connections between the
 3993 Kalman smoother and the forward-backward algorithm are elucidated by Minka (1999)
 3994 and Murphy (2012).

3995 **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):

$$p(y_m | y_{-m}, w_{1:M}) \propto p(w_m | y_m) p(y_m | y_{-m}). \quad [7.108]$$

3996 Gibbs Sampling has been applied to unsupervised part-of-speech tagging by Goldwater
 3997 and Griffiths (2007). **Beam sampling** is a more sophisticated sampling algorithm, which
 3998 randomly draws entire sequences $y_{1:M}$, rather than individual tags y_m ; this algorithm
 3999 was applied to unsupervised part-of-speech tagging by Van Gael et al. (2009). Spectral
 4000 learning (see § 5.5.2) can also be applied to sequence labeling. By factoring matrices of
 4001 co-occurrence counts of word bigrams and trigrams (Song et al., 2010; Hsu et al., 2012), it
 4002 is possible to obtain globally optimal estimates of the transition and emission parameters,
 4003 under mild assumptions.

4004 **7.7.3 Semiring notation and the generalized viterbi algorithm**

The Viterbi and Forward recurrences can each be performed over probabilities or log probabilities, yielding a total of four closely related recurrences. These four recurrence scan in fact be expressed as a single recurrence in a more general notation, known as **semiring algebra**. Let the symbols \oplus and \otimes represent generalized addition and multiplication respectively.¹¹ Given these operators, a generalized Viterbi recurrence is denoted,

$$v_m(k) = \bigoplus_{k' \in \mathcal{Y}} s_m(k, k') \otimes v_{m-1}(k'). \quad [7.109]$$

¹¹In a semiring, the addition and multiplication operators must both obey associativity, and multiplication must distribute across addition; the addition operator must be commutative; there must be additive and multiplicative identities $\bar{0}$ and $\bar{1}$, such that $a \oplus \bar{0} = a$ and $a \otimes \bar{1} = a$; and there must be a multiplicative annihilator $\bar{0}$, such that $a \otimes \bar{0} = \bar{0}$.

4005 Each recurrence that we have seen so far is a special case of this generalized Viterbi
 4006 recurrence:

- 4007 • In the max-product Viterbi recurrence over probabilities, the \oplus operation corre-
 4008 sponds to maximization, and the \otimes operation corresponds to multiplication.
- 4009 • In the forward recurrence over probabilities, the \oplus operation corresponds to addition,
 4010 and the \otimes operation corresponds to multiplication.
- 4011 • In the max-product Viterbi recurrence over log-probabilities, the \oplus operation corre-
 4012 sponds to maximization, and the \otimes operation corresponds to addition.¹²
- 4013 • In the forward recurrence over log-probabilities, the \oplus operation corresponds to log-
 4014 addition, $a \oplus b = \log(e^a + e^b)$. The \otimes operation corresponds to addition.

4015 The mathematical abstraction offered by semiring notation can be applied to the soft-
 4016 ware implementations of these algorithms, yielding concise and modular implemen-
 4017 tations. For example, in the OPENFST library, generic operations are parametrized by the
 4018 choice of semiring (Allauzen et al., 2007).

4019 Exercises

- 4020 1. Extend the example in § 7.3.1 to the sentence *they can can fish*, meaning that “they can
 4021 put fish into cans.” Build the trellis for this example using the weights in Table 7.1,
 4022 and identify the best-scoring tag sequence. If the scores for noun and verb are tied,
 4023 then you may assume that the backpointer always goes to noun.
- 4024 2. Using the tagset $\mathcal{Y} = \{N, V\}$, and the feature set $f(\mathbf{w}, y_m, y_{m-1}, m) = \{(w_m, y_m), (y_m, y_{m-1})\}$,
 4025 show that there is no set of weights that give the correct tagging for both *they can*
 4026 *fish* (N V V) and *they can can fish* (N V V N).
- 4027 3. Work out what happens if you train a structured perceptron on the two exam-
 4028 ples mentioned in the previous problem, using the transition and emission features
 4029 (y_m, y_{m-1}) and (y_m, w_m) . Initialize all weights at 0, and assume that the Viterbi algo-
 4030 rithm always chooses *N* when the scores for the two tags are tied, so that the initial
 4031 prediction for *they can fish* is N N N.
- 4032 4. Consider the garden path sentence, *The old man the boat*. Given word-tag and tag-tag
 4033 features, what inequality in the weights must hold for the correct tag sequence to
 4034 outscore the garden path tag sequence for this example?

¹²This is sometimes called the **tropical semiring**, in honor of the Brazilian mathematician Imre Simon.

- 4035 5. Using the weights in Table 7.1, explicitly compute the log-probabilities for all pos-
 4036 sible taggings of the input *fish can*. Verify that the forward algorithm recovers the
 4037 aggregate log probability.
- 4038 6. Sketch out an algorithm for a variant of Viterbi that returns the top-*n* label se-
 4039 quences. What is the time and space complexity of this algorithm?
- 4040 7. Show how to compute the marginal probability $\Pr(y_{m-2} = k, y_m = k' \mid \mathbf{w}_{1:M})$, in
 4041 terms of the forward and backward variables, and the potentials $s_n(y_n, y_{n-1})$.
- 4042 8. Suppose you receive a stream of text, where some of tokens have been replaced at
 4043 random with *NOISE*. For example:
- 4044 • Source: *I try all things, I achieve what I can*
 - 4045 • Message received: *I try NOISE NOISE, I NOISE what I NOISE*
- 4046 Assume you have access to a pre-trained bigram language model, which gives prob-
 4047 abilities $p(w_m \mid w_{m-1})$. These probabilities can be assumed to be non-zero for all
 4048 bigrams.
- 4049 Show how to use the Viterbi algorithm to recover the source by maximizing the
 4050 bigram language model log-probability. Specifically, set the scores $s_m(y_m, y_{m-1})$ so
 4051 that the Viterbi algorithm selects a sequence of words that maximizes the bigram
 4052 language model log-probability, while leaving the non-noise tokens intact. Your
 4053 solution should not modify the logic of the Viterbi algorithm, it should only set the
 4054 scores $s_m(y_m, y_{m-1})$.
- 4055 9. Let $\alpha(\cdot)$ and $\beta(\cdot)$ indicate the forward and backward variables as defined in § 7.5.3.
 4056 Prove that $\alpha_{M+1}(\blacklozenge) = \beta_0(\lozenge) = \sum_y \alpha_m(y)\beta_m(y), \forall m \in \{1, 2, \dots, M\}$.
- 4057 10. Consider an RNN tagging model with a tanh activation function on the hidden
 4058 layer, and a hinge loss on the output. (The problem also works for the margin loss
 4059 and negative log-likelihood.) Suppose you initialize all parameters to zero: this in-
 4060 cludes the word embeddings that make up \mathbf{x} , the transition matrix Θ , the output
 4061 weights β , and the initial hidden state \mathbf{h}_0 .
 - 4062 a) Prove that for any data and for any gradient-based learning algorithm, all pa-
 4063 rameters will be stuck at zero.
 - 4064 b) Would a sigmoid activation function avoid this problem?

4065

Chapter 8

4066

Applications of sequence labeling

4067 Sequence labeling has applications throughout natural language processing. This chapter
4068 focuses on part-of-speech tagging, morpho-syntactic attribute tagging, named entity
4069 recognition, and tokenization. It also touches briefly on two applications to interactive
4070 settings: dialogue act recognition and the detection of code-switching points between
4071 languages.

4072

8.1 Part-of-speech tagging

4073 The **syntax** of a language is the set of principles under which sequences of words are
4074 judged to be grammatically acceptable by fluent speakers. One of the most basic syntactic
4075 concepts is the **part-of-speech** (POS), which refers to the syntactic role of each word in a
4076 sentence. This concept was used informally in the previous chapter, and you may have
4077 some intuitions from your own study of English. For example, in the sentence *We like*
4078 *vegetarian sandwiches*, you may already know that *we* and *sandwiches* are nouns, *like* is a
4079 verb, and *vegetarian* is an adjective. These labels depend on the context in which the word
4080 appears: in *she eats like a vegetarian*, the word *like* is a preposition, and the word *vegetarian*
4081 is a noun.

4082 Parts-of-speech can help to disentangle or explain various linguistic problems. Recall
4083 Chomsky's proposed distinction in chapter 6:

- 4084 (8.1) a. Colorless green ideas sleep furiously.
4085 b. * Ideas colorless furiously green sleep.

4086 One difference between these two examples is that the first contains part-of-speech trans-
4087 sitions that are typical in English: adjective to adjective, adjective to noun, noun to verb,
4088 and verb to adverb. The second example contains transitions that are unusual: noun to
4089 adjective and adjective to verb. The ambiguity in a headline like,

4090 (8.2) Teacher Strikes Idle Children

4091 can also be explained in terms of parts of speech: in the interpretation that was likely
 4092 intended, *strikes* is a noun and *idle* is a verb; in the alternative explanation, *strikes* is a verb
 4093 and *idle* is an adjective.

4094 Part-of-speech tagging is often taken as a early step in a natural language processing
 4095 pipeline. Indeed, parts-of-speech provide features that can be useful for many of the
 4096 tasks that we will encounter later, such as parsing (chapter 10), coreference resolution
 4097 (chapter 15), and relation extraction (chapter 17).

4098 **8.1.1 Parts-of-Speech**

4099 The **Universal Dependencies** project (UD) is an effort to create syntactically-annotated
 4100 corpora across many languages, using a single annotation standard (Nivre et al., 2016). As
 4101 part of this effort, they have designed a part-of-speech **tagset**, which is meant to capture
 4102 word classes across as many languages as possible.¹ This section describes that inventory,
 4103 giving rough definitions for each of tags, along with supporting examples.

4104 Part-of-speech tags are **morphosyntactic**, rather than semantic, categories. This means
 4105 that they describe words in terms of how they pattern together and how they are inter-
 4106 nally constructed (e.g., what suffixes and prefixes they include). For example, you may
 4107 think of a noun as referring to objects or concepts, and verbs as referring to actions or
 4108 events. But events can also be nouns:

4109 (8.3) ... the **howling** of the **shrieking** storm.

4110 Here *howling* and *shrieking* are events, but grammatically they act as a noun and adjective
 4111 respectively.

4112 **The Universal Dependency part-of-speech tagset**

4113 The UD tagset is broken up into three groups: open class tags, closed class tags, and
 4114 “others.”

4115 **Open class tags** Nearly all languages contain nouns, verbs, adjectives, and adverbs.²
 4116 These are all **open word classes**, because new words can easily be added to them. The
 4117 UD tagset includes two other tags that are open classes: proper nouns and interjections.

4118 • **Nouns** (UD tag: NOUN) tend to describe entities and concepts, e.g.,

¹The UD tagset builds on earlier work from Petrov et al. (2012), in which a set of twelve universal tags was identified by creating mappings from tagsets for individual languages.

²One prominent exception is Korean, which some linguists argue does not have adjectives Kim (2002).

4119 (8.4) **Toes** are scarce among veteran **blubber men**.

4120 In English, nouns tend to follow determiners and adjectives, and can play the subject
4121 role in the sentence. They can be marked for the plural number by an *-s* suffix.

- 4122 • **Proper nouns** (PROPN) are tokens in names, which uniquely specify a given entity,

4123 (8.5) “**Moby Dick?**” shouted **Ahab**.

- 4124 • **Verbs** (VERB), according to the UD guidelines, “typically signal events and ac-
4125 tions.” But they are also defined grammatically: they “can constitute a minimal
4126 predicate in a clause, and govern the number and types of other constituents which
4127 may occur in a clause.”³

4128 (8.6) “**Moby Dick?**” shouted Ahab.

4129 (8.7) Shall we **keep chasing** this murderous fish?

4130 English verbs tend to come in between the subject and some number of direct ob-
4131 jects, depending on the verb. They can be marked for **tense** and **aspect** using suffixes
4132 such as *-ed* and *-ing*. (These suffixes are an example of **inflectional morphology**,
4133 which is discussed in more detail in § 9.1.4.)

- 4134 • **Adjectives** (ADJ) describe properties of entities,

4135 (8.8) a. Shall we keep chasing this **murderous** fish?

4136 b. Toes are **scarce** among **veteran** blubber men.

4137 In the second example, *scarce* is a predicative adjective, linked to the subject by the
4138 **copula verb** *are*. In contrast, *murderous* and *veteran* are attributive adjectives, modi-
4139 fying the noun phrase in which they are embedded.

- 4140 • **Adverbs** (ADV) describe properties of events, and may also modify adjectives or
4141 other adverbs:

4142 (8.9) a. It is not down on any map; true places **never** are.

4143 b. ...**treacherously** hidden beneath the loveliest tints of azure

4144 c. Not drowned **entirely**, though.

- 4145 • **Interjections** (INTJ) are used in exclamations, e.g.,

4146 (8.10) **Aye aye!** it was that accursed white whale that razed me.

³<http://universaldependencies.org/u/pos/VERB.html>

4147 **Closed class tags** Closed word classes rarely receive new members. They are sometimes
 4148 referred to as **function words** — as opposed to **content words** — as they have little lexical
 4149 meaning of their own, but rather, help to organize the components of the sentence.

4150 • **Adpositions** (ADP) describe the relationship between a complement (usually a noun
 4151 phrase) and another unit in the sentence, typically a noun or verb phrase.

- 4152 (8.11) a. Toes are scarce **among** veteran blubber men.
 4153 b. It is not **down on** any map.
 4154 c. Give not thyself **up** then.

4155 As the examples show, English generally uses prepositions, which are adpositions
 4156 that appear before their complement. (An exception is *ago*, as in, *we met three days*
 4157 *ago*). Postpositions are used in other languages, such as Japanese and Turkish.

4158 • **Auxiliary verbs** (AUX) are a closed class of verbs that add information such as
 4159 tense, aspect, person, and number.

- 4160 (8.12) a. **Shall** we keep chasing this murderous fish?
 4161 b. What the white whale was to Ahab, **has been** hinted.
 4162 c. Ahab **must** use tools.
 4163 d. Meditation and water **are** wedded forever.
 4164 e. Toes **are** scarce among veteran blubber men.

4165 The final example is a copula verb, which is also tagged as an auxiliary in the UD
 4166 corpus.

4167 • **Coordinating conjunctions** (CCONJ) express relationships between two words or
 4168 phrases, which play a parallel role:

- 4169 (8.13) Meditation **and** water are wedded forever.

4170 • **Subordinating conjunctions** (SCONJ) link two clauses, making one syntactically
 4171 subordinate to the other:

- 4172 (8.14) It is the easiest thing in the world for a man to look as **if** he had a great
 4173 secret in him.

4174 Note that

4175 • **Pronouns** (PRON) are words that substitute for nouns or noun phrases.

- 4176 (8.15) a. Be **it what it will**, I'll go to **it** laughing.

4177 b. I try all things, I achieve **what** I can.

4178 The example includes the personal pronouns *I* and *it*, as well as the relative pronoun
4179 *what*. Other pronouns include *myself*, *somebody*, and *nothing*.

- 4180 • **Determiners** (DET) provide additional information about the nouns or noun phrases
4181 that they modify:

- 4182 (8.16) a. What **the** white whale was to Ahab, has been hinted.
4183 b. It is not down on **any** map.
4184 c. I try **all** things ...
4185 d. Shall we keep chasing **this** murderous fish?

4186 Determiners include articles (*the*), possessive determiners (*their*), demonstratives
4187 (*this murderous fish*), and quantifiers (*any map*).

- 4188 • **Numerals** (NUM) are an infinite but closed class, which includes integers, fractions,
4189 and decimals, regardless of whether spelled out or written in numerical form.

- 4190 (8.17) a. How then can this **one** small heart beat.
4191 b. I am going to put him down for the **three hundredth**.

- 4192 • **Particles** (PART) are a catch-all of function words that combine with other words or
4193 phrases, but do not meet the conditions of the other tags. In English, this includes
4194 the infinitival *to*, the possessive marker, and negation.

- 4195 (8.18) a. Better **to** sleep with a sober cannibal than a drunk Christian.
4196 b. So man's insanity is heaven's sense
4197 c. It is **not** down on any map

4198 As the second example shows, the possessive marker is not considered part of the
4199 same token as the word that it modifies, so that *man's* is split into two tokens. (Tok-
4200 enization is described in more detail in § 8.4.) A non-English example of a particle
4201 is the Japanese question marker *ka*:⁴

- 4202 (8.19) Sensei desu ka
4203 Teacher is ?
 Is she a teacher?

⁴In this notation, the first line is the transliterated Japanese text, the second line is a token-to-token **gloss**, and the third line is the translation.

4204 **Other** The remaining UD tags include punctuation (PUN) and symbols (SYM). Punc-
 4205 tuation is purely structural — e.g., commas, periods, colons — while symbols can carry
 4206 content of their own. Examples of symbols include dollar and percentage symbols, math-
 4207 ematical operators, emoticons, emojis, and internet addresses. A final catch-all tag is X,
 4208 which is used for words that cannot be assigned another part-of-speech category. The X
 4209 tag is also used in cases of **code switching** (between languages), described in § 8.5.

4210 **Other tagsets**

4211 Prior to the Universal Dependency treebank, part-of-speech tagging was performed us-
 4212 ing language-specific tagsets. The dominant tagset for English was designed as part of
 4213 the **Penn Treebank** (PTB), and it includes 45 tags — more than three times as many as
 4214 the UD tagset. This granularity is reflected in distinctions between singular and plural
 4215 nouns, verb tenses and aspects, possessive and non-possessive pronouns, comparative
 4216 and superlative adjectives and adverbs (e.g., *faster, fastest*), and so on. The Brown corpus
 4217 includes a tagset that is even more detailed, with 87 tags (Francis, 1964), including special
 4218 tags for individual auxiliary verbs such as *be, do, and have*.

4219 Different languages make different distinctions, and so the PTB and Brown tagsets are
 4220 not appropriate for a language such as Chinese, which does not mark the verb tense (Xia,
 4221 2000); nor for Spanish, which marks every combination of person and number in the
 4222 verb ending; nor for German, which marks the case of each noun phrase. Each of these
 4223 languages requires more detail than English in some areas of the tagset, and less in other
 4224 areas. The strategy of the Universal Dependencies corpus is to design a coarse-grained
 4225 tagset to be used across all languages, and then to additionally annotate language-specific
 4226 **morphosyntactic attributes**, such as number, tense, and case. The attribute tagging task
 4227 is described in more detail in § 8.2.

4228 Social media such as Twitter have been shown to require tagsets of their own (Gimpel
 4229 et al., 2011). Such corpora contain some tokens that are not equivalent to anything en-
 4230 countered in a typical written corpus: e.g., emoticons, URLs, and hashtags. Social media
 4231 also includes dialectal words like *gonna* ('going to', e.g. *We gonna be fine*) and *Ima* ('I'm
 4232 going to', e.g., *Ima tell you one more time*), which can be analyzed either as non-standard
 4233 orthography (making tokenization impossible), or as lexical items in their own right. In
 4234 either case, it is clear that existing tags like NOUN and VERB cannot handle cases like *Ima*,
 4235 which combine aspects of the noun and verb. Gimpel et al. (2011) therefore propose a new
 4236 set of tags to deal with these cases.

4237 **8.1.2 Accurate part-of-speech tagging**

4238 Part-of-speech tagging is the problem of selecting the correct tag for each word in a sen-
 4239 tence. Success is typically measured by accuracy on an annotated test set, which is simply
 4240 the fraction of tokens that were tagged correctly.

4241 **Baselines**

4242 A simple baseline for part-of-speech tagging is to choose the most common tag for each
4243 word. For example, in the Universal Dependencies treebank, the word *talk* appears 96
4244 times, and 85 of those times it is labeled as a VERB: therefore, this baseline will always
4245 predict VERB for this word. For words that do not appear in the training corpus, the base-
4246 line simply guesses the most common tag overall, which is NOUN. In the Penn Treebank,
4247 this simple baseline obtains accuracy above 92%. A more rigorous evaluation is the accu-
4248 racy on **out-of-vocabulary words**, which are not seen in the training data. Tagging these
4249 words correctly requires attention to the context and the word's internal structure.

4250 **Contemporary approaches**

4251 Conditional random fields and structured perceptron perform at or near the state-of-the-
4252 art for part-of-speech tagging in English. For example, (Collins, 2002) achieved 97.1%
4253 accuracy on the Penn Treebank, using a structured perceptron with the following base
4254 features (originally introduced by Ratnaparkhi (1996)):

- 4255 • current word, w_m
- 4256 • previous words, w_{m-1}, w_{m-2}
- 4257 • next words, w_{m+1}, w_{m+2}
- 4258 • previous tag, y_{m-1}
- 4259 • previous two tags, (y_{m-1}, y_{m-2})
- 4260 • for rare words:
 - 4261 – first k characters, up to $k = 4$
 - 4262 – last k characters, up to $k = 4$
 - 4263 – whether w_m contains a number, uppercase character, or hyphen.

4264 Similar results for the PTB data have been achieved using conditional random fields (CRFs;
4265 Toutanova et al., 2003).

4266 More recent work has demonstrated the power of neural sequence models, such as the
4267 **long short-term memory (LSTM)** (§ 7.6). Plank et al. (2016) apply a CRF and a bidirec-
4268 tional LSTM to twenty-two languages in the UD corpus, achieving an average accuracy
4269 of 94.3% for the CRF, and 96.5% with the bi-LSTM. Their neural model employs three
4270 types of embeddings: fine-tuned word embeddings, which are updated during training;
4271 pre-trained word embeddings, which are never updated, but which help to tag out-of-
4272 vocabulary words; and character-based embeddings. The character-based embeddings
4273 are computed by running an LSTM on the individual characters in each word, thereby
4274 capturing common orthographic patterns such as prefixes, suffixes, and capitalization.
4275 Extensive evaluations show that these additional embeddings are crucial to their model's
4276 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.

4277 8.2 Morphosyntactic Attributes

4278 There is considerably more to say about a word than whether it is a noun or a verb: in En-
 4279 glish, verbs are distinguish by features such tense and aspect, nouns by number, adjectives
 4280 by degree, and so on. These features are language-specific: other languages distinguish
 4281 other features, such as **case** (the role of the noun with respect to the action of the sen-
 4282 tence, which is marked in languages such as Latin and German⁵) and **evidentiality** (the
 4283 source of information for the speaker’s statement, which is marked in languages such as
 4284 Turkish). In the UD corpora, these attributes are annotated as feature-value pairs for each
 4285 token.⁶

4286 An example is shown in Figure 8.1. The determiner *the* is marked with two attributes:
 4287 PRONTYPE=ART, which indicates that it is an **article** (as opposed to another type of deter-

⁵Case is marked in English for some personal pronouns, e.g., *She saw her, They saw them*.

⁶The annotation and tagging of morphosyntactic attributes can be traced back to earlier work on Turkish (Oflazer and Kuruöz, 1994) and Czech (Hajič and Hladká, 1998). MULTEXT-East was an early multilingual corpus to include morphosyntactic attributes (Dimitrova et al., 1998).

miner or pronominal modifier), and DEFINITE=DEF, which indicates that it is a **definite article** (referring to a specific, known entity). The verbs are each marked with several attributes. The auxiliary verb *was* is third-person, singular, past tense, finite (conjugated), and indicative (describing an event that has happened or is currently happenings); the main verb *destroyed* is in participle form (so there is no additional person and number information), past tense, and passive voice. Some, but not all, of these distinctions are reflected in the PTB tags VBD (past-tense verb) and VBN (past participle).

While there are thousands of papers on part-of-speech tagging, there is comparatively little work on automatically labeling morphosyntactic attributes. Faruqui et al. (2016) train a support vector machine classification model, using a minimal feature set that includes the word itself, its prefixes and suffixes, and type-level information listing all possible morphosyntactic attributes for each word and its neighbors. Mueller et al. (2013) use a conditional random field (CRF), in which the tag space consists of all observed combinations of morphosyntactic attributes (e.g., the tag would be DEF+ART for the word *the* in Figure 8.1). This massive tag space is managed by decomposing the feature space over individual attributes, and pruning paths through the trellis. More recent work has employed bidirectional LSTM sequence models. For example, Pinter et al. (2017) train a bidirectional LSTM sequence model. The input layer and hidden vectors in the LSTM are shared across attributes, but each attribute has its own output layer, culminating in a softmax over all attribute values, e.g. $y_t^{\text{NUMBER}} \in \{\text{SING}, \text{PLURAL}, \dots\}$. They find that character-level information is crucial, especially when the amount of labeled data is limited.

Evaluation is performed by first computing recall and precision for each attribute. These scores can then be averaged at either the type or token level to obtain micro- or macro-*F*-MEASURE. Pinter et al. (2017) evaluate on 23 languages in the UD treebank, reporting a median micro-*F*-MEASURE of 0.95. Performance is strongly correlated with the size of the labeled dataset for each language, with a few outliers: for example, Chinese is particularly difficult, because although the dataset is relatively large (10^5 tokens in the UD 1.4 corpus), only 6% of tokens have any attributes, offering few useful labeled instances.

8.3 Named Entity Recognition

A classical problem in information extraction is to recognize and extract mentions of **named entities** in text. In news documents, the core entity types are people, locations, and organizations; more recently, the task has been extended to include amounts of money, percentages, dates, and times. In item 8.20a (Figure 8.2), the named entities include: *The U.S. Army*, an organization; *Atlanta*, a location; and *May 14, 1864*, a date. Named entity recognition is also a key task in **biomedical natural language processing**, with entity types including proteins, DNA, RNA, and cell lines (e.g., Collier et al., 2000; Ohta et al., 2002). Figure 8.2 shows an example from the GENIA corpus of biomedical research ab-

- (8.20) a. *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
 b. *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 (8.20b) is drawn from the GENIA corpus of biomedical documents (Ohta et al., 2002).

4326 stracts.

4327 A standard approach to tagging named entity spans is to use discriminative sequence
 4328 labeling methods such as conditional random fields. However, the named entity recogni-
 4329 tion (NER) task would seem to be fundamentally different from sequence labeling tasks
 4330 like part-of-speech tagging: rather than tagging each token, the goal is to recover *spans*
 4331 of tokens, such as *The United States Army*.

4332 This is accomplished by the **BIO notation**, shown in Figure 8.2. Each token at the
 4333 beginning of a name span is labeled with a B- prefix; each token within a name span is la-
 4334 beled with an I- prefix. These prefixes are followed by a tag for the entity type, e.g. B-LOC
 4335 for the beginning of a location, and I-PROTEIN for the inside of a protein name. Tokens
 4336 that are not parts of name spans are labeled as O. From this representation, the entity
 4337 name spans can be recovered unambiguously. This tagging scheme is also advantageous
 4338 for learning: tokens at the beginning of name spans may have different properties than
 4339 tokens within the name, and the learner can exploit this. This insight can be taken even
 4340 further, with special labels for the last tokens of a name span, and for unique tokens in
 4341 name spans, such as *Atlanta* in the example in Figure 8.2. This is called BILOU notation,
 4342 and it can yield improvements in supervised named entity recognition (Ratinov and Roth,
 4343 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 (8.20a) 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*)

4344 Features can also be obtained from a **gazetteer**, which is a list of known entity names. For
 4345 example, the U.S. Social Security Administration provides a list of tens of thousands of

- (1) 日文 章魚 怎麼 說?
 Japanese octopus how say
 How to say octopus in Japanese?
- (2) 日 文章 魚 怎麼 說?
 Japan essay fish how say

Figure 8.3: An example of tokenization ambiguity in Chinese (Sproat et al., 1996)

4346 given names — more than could be observed in any annotated corpus. Tokens or spans
 4347 that match an entry in a gazetteer can receive special features; this provides a way to
 4348 incorporate hand-crafted resources such as name lists in a learning-driven framework.

4349 **Neural sequence labeling for NER** Current research has emphasized neural sequence
 4350 labeling, using similar LSTM models to those employed in part-of-speech tagging (Ham-
 4351 merton, 2003; Huang et al., 2015; Lample et al., 2016). The bidirectional LSTM-CRF (Fig-
 4352 ure 7.4 in § 7.6) does particularly well on this task, due to its ability to model tag-to-tag
 4353 dependencies. However, Strubell et al. (2017) show that **convolutional neural networks**
 4354 can be equally accurate, with significant improvement in speed due to the efficiency of
 4355 implementing ConvNets on **graphics processing units (GPUs)**. The key innovation in
 4356 this work was the use of **dilated convolution**, which is described in more detail in § 3.4.

4357 8.4 Tokenization

4358 A basic problem for text analysis, first discussed in § 4.3.1, is to break the text into a se-
 4359 quence of discrete tokens. For alphabetic languages such as English, deterministic scripts
 4360 usually suffice to achieve accurate tokenization. However, in logographic writing systems
 4361 such as Chinese script, words are typically composed of a small number of characters,
 4362 without intervening whitespace. The tokenization must be determined by the reader, with
 4363 the potential for occasional ambiguity, as shown in Figure 8.3. One approach is to match
 4364 character sequences against a known dictionary (e.g., Sproat et al., 1996), using additional
 4365 statistical information about word frequency. However, no dictionary is completely com-
 4366 prehensive, and dictionary-based approaches can struggle with such out-of-vocabulary
 4367 words.

4368 Chinese word segmentation has therefore been approached as a supervised sequence
 4369 labeling problem. Xue et al. (2003) train a logistic regression classifier to make indepen-
 4370 dent segmentation decisions while moving a sliding window across the document. A set
 4371 of rules is then used to convert these individual classification decisions into an overall to-
 4372 kenization of the input. However, these individual decisions may be globally suboptimal,
 4373 motivating a structure prediction approach. Peng et al. (2004) train a conditional random

4374 field to predict labels of START or NONSTART on each character. More recent work has
 4375 employed neural network architectures. For example, Chen et al. (2015) use an LSTM-
 4376 CRF architecture, as described in § 7.6: they construct a trellis, in which each tag is scored
 4377 according to the hidden state of an LSTM, and tag-tag transitions are scored according
 4378 to learned transition weights. The best-scoring segmentation is then computed by the
 4379 Viterbi algorithm.

4380 8.5 Code switching

4381 Multilingual speakers and writers do not restrict themselves to a single language. **Code**
4382 **switching** is the phenomenon of switching between languages in speech and text (Auer,
4383 2013; Poplack, 1980). Written code switching has become more common in online social
4384 media, as in the following extract from the website of Canadian President Justin Trudeau:⁷

- 4385 (8.21) *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*

4387 Accurately analyzing such texts requires first determining which languages are being
4388 used. Furthermore, quantitative analysis of code switching can provide insights on the
4389 languages themselves and their relative social positions.

Code switching can be viewed as a sequence labeling problem, where the goal is to label each token as a candidate switch point. In the example above, the words *est*, *and*, and *seront* would be labeled as switch points. Solorio and Liu (2008) detect English-Spanish switch points using a supervised classifier, with features that include the word, its part-of-speech in each language (according to a supervised part-of-speech tagger), and the probabilities of the word and part-of-speech in each language. Nguyen and Dogruöz (2013) apply a conditional random field to the problem of detecting code switching between Turkish and Dutch.

Code switching is a special case of the more general problem of word level language identification, which Barman et al. (2014) address in the context of trilingual code switching between Bengali, English, and Hindi. They further observe an even more challenging phenomenon: intra-word code switching, such as the use of English suffixes with Bengali roots. They therefore mark each token as either (1) belonging to one of the three languages; (2) a mix of multiple languages; (3) “universal” (e.g., symbols, numbers, emoticons); or (4) undefined.

⁷As quoted in <http://blogues.lapresse.ca/lagace/2008/09/08/justin-trudeau-really-parfait-bilingue/>, accessed August 21, 2017.

| Speaker | Dialogue Act | Utterance |
|---------|----------------------|---|
| A | YES-NO-QUESTION | <i>So do you go college right now?</i> |
| A | ABANDONED | <i>Are yo-</i> |
| B | YES-ANSWER | <i>Yeah,</i> |
| B | STATEMENT | <i>It's my last year [laughter].</i> |
| A | DECLARATIVE-QUESTION | <i>You're a, so you're a senior now.</i> |
| B | YES-ANSWER | <i>Yeah,</i> |
| B | STATEMENT | <i>I'm working on my projects trying to graduate [laughter]</i> |
| A | APPRECIATION | <i>Oh, good for you.</i> |
| B | BACKCHANNEL | <i>Yeah.</i> |

Figure 8.4: An example of dialogue act labeling (Stolcke et al., 2000)

4405 8.6 Dialogue acts

4406 The sequence labeling problems that we have discussed so far have been over sequences
 4407 of word tokens or characters (in the case of tokenization). However, sequence labeling
 4408 can also be performed over higher-level units, such as **utterances**. **Dialogue acts** are la-
 4409 bels over utterances in a dialogue, corresponding roughly to the speaker’s intention —
 4410 the utterance’s **illocutionary force** (Austin, 1962). For example, an utterance may state a
 4411 proposition (*it is not down on any map*), pose a question (*shall we keep chasing this murderous*
 4412 *fish?*), or provide a response (*aye aye!*). Stolcke et al. (2000) describe how a set of 42 dia-
 4413 logue acts were annotated for the 1,155 conversations in the Switchboard corpus (Godfrey
 4414 et al., 1992).⁸

4415 An example is shown in Figure 8.4. The annotation is performed over UTTERANCES,
 4416 with the possibility of multiple utterances per **conversational turn** (in cases such as inter-
 4417 ruptions, an utterance may split over multiple turns). Some utterances are clauses (e.g., *So*
 4418 *do you go to college right now?*), while others are single words (e.g., *yeah*). Stolcke et al. (2000)
 4419 report that hidden Markov models (HMMs) achieve 96% accuracy on supervised utter-
 4420 ance segmentation. The labels themselves reflect the conversational goals of the speaker:
 4421 the utterance *yeah* functions as an answer in response to the question *you’re a senior now*,
 4422 but in the final line of the excerpt, it is a **backchannel** (demonstrating comprehension).

4423 For task of dialogue act labeling, Stolcke et al. (2000) apply a hidden Markov model.
 4424 The probability $p(w_m | y_m)$ must generate the entire sequence of words in the utterance,
 4425 and it is modeled as a trigram language model (§ 6.1). Stolcke et al. (2000) also account
 4426 for acoustic features, which capture the **prosody** of each utterance — for example, tonal
 4427 and rhythmic properties of speech, which can be used to distinguish dialogue acts such

⁸Dialogue act modeling is not restricted to speech; it is relevant in any interactive conversation. For example, Jeong et al. (2009) annotate a more limited set of **speech acts** in a corpus of emails and online forums.

4428 as questions and answers. These features are handled with an additional emission distri-
 4429 bution, $p(a_m | y_m)$, which is modeled with a probabilistic decision tree (Murphy, 2012).
 4430 While acoustic features yield small improvements overall, they play an important role in
 4431 distinguish questions from statements, and agreements from backchannels.

4432 Recurrent neural architectures for dialogue act labeling have been proposed by Kalch-
 4433 brenner and Blunsom (2013) and Ji et al. (2016), with strong empirical results. Both models
 4434 are recurrent at the utterance level, so that each complete utterance updates a hidden state.
 4435 The recurrent-convolutional network of Kalchbrenner and Blunsom (2013) uses convolu-
 4436 tion to obtain a representation of each individual utterance, while Ji et al. (2016) use a
 4437 second level of recurrence, over individual words. This enables their method to also func-
 4438 tion as a language model, giving probabilities over sequences of words in a document.

4439 Exercises

4440 1. Using the Universal Dependencies part-of-speech tags, annotate the following sen-
 4441 tences. You may examine the UD tagging guidelines. Tokenization is shown with
 4442 whitespace. Don't forget about punctuation.

- 4443 (8.22) a. I try all things , I achieve what I can .
 4444 b. It was that accursed white whale that razed me .
 4445 c. Better to sleep with a sober cannibal , than a drunk Christian .
 4446 d. Be it what it will , I 'll go to it laughing .

4447 2. Select three short sentences from a recent news article, and annotate them for UD
 4448 part-of-speech tags. Ask a friend to annotate the same three sentences without look-
 4449 ing at your annotations. Compute the rate of agreement, using the Kappa metric
 4450 defined in § 4.5.2. Then work together to resolve any disagreements.

- 4451 3. Choose one of the following morphosyntactic attributes: MOOD, TENSE, VOICE. Re-
 4452 search the definition of this attribute on the universal dependencies website, <http://universaldependencies.org/u/feat/index.html>. Returning to the ex-
 4453 amples in the first exercise, annotate all verbs for your chosen attribute. It may be
 4454 helpful to consult examples from an English-language universal dependencies cor-
 4455 pus, available at [https://github.com/UniversalDependencies/UD_English-EWT/
 4456 tree/master](https://github.com/UniversalDependencies/UD_English-EWT/tree/master).
- 4458 4. Download a dataset annotated for universal dependencies, such as the English Tree-
 4459 bank at [https://github.com/UniversalDependencies/UD_English-EWT/
 4460 tree/master](https://github.com/UniversalDependencies/UD_English-EWT/tree/master). This corpus is already segmented into training, development, and
 4461 test data.

- 4462 a) First, train a logistic regression or SVM classifier using character suffixes: char-
4463 acter n-grams up to length 4. Compute the recall, precision, and *F*-MEASURE
4464 on the development data.
- 4465 b) Next, augment your classifier using the same character suffixes of the preced-
4466 ing and succeeding tokens. Again, evaluate your classifier on heldout data.
- 4467 c) Optionally, train a Viterbi-based sequence labeling model, using a toolkit such
4468 as CRFSuite (<http://www.chokkan.org/software/crfsuite/>) or your
4469 own Viterbi implementation. This is more likely to be helpful for attributes
4470 in which agreement is required between adjacent words. For example, many
4471 Romance languages require gender and number agreement for determiners,
4472 nouns, and adjectives.
- 4473 5. Provide BIO-style annotation of the named entities (person, place, organization,
4474 date, or product) in the following expressions:
- 4475 (8.23) a. The third mate was Flask, a native of Tisbury, in Martha's Vineyard.
4476 b. Its official Nintendo announced today that they Will release the Nin-
4477 tendo 3DS in north America march 27 (Ritter et al., 2011).
4478 c. Jessica Reif, a media analyst at Merrill Lynch & Co., said, "If they can
4479 get up and running with exclusive programming within six months, it
4480 doesn't set the venture back that far."⁹
- 4481 6. Run the examples above through the online version of a named entity recogni-
4482 tion tagger, such as the Allen NLP system here: <http://demo.allennlp.org/named->
4483 entity-recognition. Do the predicted tags match your annotations?
- 4484 7. Build a whitespace tokenizer for English:
- 4485 a) Using the NLTK library, download the complete text to the novel *Alice in Won-*
4486 *derland* (Carroll, 1865). Hold out the final 1000 words as a test set.
- 4487 b) Label each alphanumeric character as a segmentation point, $y_m = 1$ if m is
4488 the final character of a token. Label every other character as $y_m = 0$. Then
4489 concatenate all the tokens in the training and test sets. Make sure that the num-
4490 ber of labels $\{y_m\}_{m=1}^M$ is identical to the number of characters $\{c_m\}_{m=1}^M$ in your
4491 concatenated datasets.
- 4492 c) Train a logistic regression classifier to predict y_m , using the surrounding char-
4493 acters $c_{m-5:m+5}$ as features. After training the classifier, run it on the test set,
4494 using the predicted segmentation points to re-tokenize the text.

⁹From the Message Understanding Conference (MUC-7) dataset (Chinchor and Robinson, 1997).

- 4495 d) Compute the per-character segmentation accuracy on the test set. You should
4496 be able to get at least 88% accuracy.
4497 e) Print out a sample of segmented text from the test set, e.g.

4498 Thereareno mice in the air , I ' m afraid , but y oumight cat
4499 chabat , and that ' s very like a mouse , youknow . But
4500 docatseat bats , I wonder ?'

- 4501 8. Perform the following extensions to your tokenizer in the previous problem.

- 4502 a) Train a conditional random field sequence labeler, by incorporating the tag
4503 bigrams (y_{m-1}, y_m) as additional features. You may use a structured predic-
4504 tion library such as CRFSuite, or you may want to implement Viterbi yourself.
4505 Compare the accuracy with your classification-based approach.
- 4506 b) Compute the token-level performance: treating the original tokenization as
4507 ground truth, compute the number of true positives (tokens that are in both
4508 the ground truth and predicted tokenization), false positives (tokens that are in
4509 the predicted tokenization but not the ground truth), and false negatives (to-
4510 kens that are in the ground truth but not the predicted tokenization). Compute
4511 the F-measure.
4512 Hint: to match predicted and ground truth tokens, add “anchors” for the start
4513 character of each token. The number of true positives is then the size of the
4514 intersection of the sets of predicted and ground truth tokens.
- 4515 c) Apply the same methodology in a more practical setting: tokenization of Chi-
4516 nese, which is written without whitespace. You can find annotated datasets at
4517 <http://alias-i.com/lingpipe/demos/tutorial/chineseTokens/read-me.html>.

4519 **Chapter 9**

4520 **Formal language theory**

4521 We have now seen methods for learning to label individual words, vectors of word counts,
4522 and sequences of words; we will soon proceed to more complex structural transfor-
4523 mations. Most of these techniques could apply to counts or sequences from any discrete vo-
4524 cabulary; there is nothing fundamentally linguistic about, say, a hidden Markov model.
4525 This raises a basic question that this text has not yet considered: what is a language?

4526 This chapter will take the perspective of **formal language theory**, in which a language
4527 is defined as a set of **strings**, each of which is a sequence of elements from a finite alphabet.
4528 For interesting languages, there are an infinite number of strings that are in the language,
4529 and an infinite number of strings that are not. For example:

- 4530 • the set of all even-length sequences from the alphabet $\{a, b\}$, e.g., $\{\emptyset, aa, ab, ba, bb, aaaa, aaab, \dots\}$;
4531 • the set of all sequences from the alphabet $\{a, b\}$ that contain *aaa* as a substring, e.g.,
4532 $\{aaa, aaaa, baaa, aaab, \dots\}$;
4533 • the set of all sequences of English words (drawn from a finite dictionary) that con-
4534 tain at least one verb (a finite subset of the dictionary);
4535 • the PYTHON programming language.

4536 Formal language theory defines classes of languages and their computational prop-
4537 erties. Of particular interest is the computational complexity of solving the **membership**
4538 **problem** — determining whether a string is in a language. The chapter will focus on
4539 three classes of formal languages: regular, context-free, and “mildly” context-sensitive
4540 languages.

4541 A key insight of 20th century linguistics is that formal language theory can be usefully
4542 applied to natural languages such as English, by designing formal languages that cap-
4543 ture as many properties of the natural language as possible. For many such formalisms, a
4544 useful linguistic analysis comes as a byproduct of solving the membership problem. The

4545 membership problem can be generalized to the problems of *scoring* strings for their ac-
 4546 ceptability (as in language modeling), and of **transducing** one string into another (as in
 4547 translation).

4548 9.1 Regular languages

4549 If you have written a **regular expression**, then you have defined a **regular language**: a
 4550 regular language is any language that can be defined by a regular expression. Formally, a
 4551 regular expression can include the following elements:

- 4552 • A **literal character** drawn from some finite alphabet Σ .
- 4553 • The **empty string** ϵ .
- 4554 • The concatenation of two regular expressions RS , where R and S are both regular
 4555 expressions. The resulting expression accepts any string that can be decomposed
 4556 $x = yz$, where y is accepted by R and z is accepted by S .
- 4557 • The alternation $R \mid S$, where R and S are both regular expressions. The resulting
 4558 expression accepts a string x if it is accepted by R or it is accepted by S .
- 4559 • The **Kleene star** R^* , which accepts any string x that can be decomposed into a se-
 4560 quence of strings which are all accepted by R .
- 4561 • Parenthesization ((R)), which is used to limit the scope of the concatenation, alterna-
 4562 tion, and Kleene star operators.

4563 Here are some example regular expressions:

- 4564 • The set of all even length strings on the alphabet $\{a, b\}$: $((aa)|(ab)|(ba)|(bb))^*$
- 4565 • The set of all sequences of the alphabet $\{a, b\}$ that contain aaa as a substring: $(a|b)^*aaa(a|b)^*$
- 4566 • The set of all sequences of English words that contain at least one verb: W^*VW^* ,
 4567 where W is an alternation between all words in the dictionary, and V is an alterna-
 4568 tion between all verbs ($V \subseteq W$).

4569 This list does not include a regular expression for the Python programming language,
 4570 because this language is not regular — there is no regular expression that can capture its
 4571 syntax. We will discuss why towards the end of this section.

4572 Regular languages are **closed** under union, intersection, and concatenation. This means
 4573 that if two languages L_1 and L_2 are regular, then so are the languages $L_1 \cup L_2$, $L_1 \cap L_2$,
 4574 and the language of strings that can be decomposed as $s = tu$, with $s \in L_1$ and $t \in L_2$.
 4575 Regular languages are also closed under negation: if L is regular, then so is the language
 4576 $\bar{L} = \{s \notin L\}$.

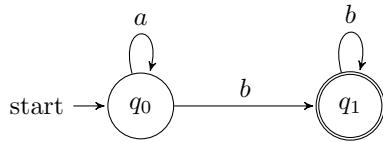


Figure 9.1: State diagram for the finite state acceptor M_1 .

9.1.1 Finite state acceptors

A regular expression defines a regular language, but does not give an algorithm for determining whether a string is in the language that it defines. **Finite state automata** are theoretical models of computation on regular languages, which involve transitions between a finite number of states. The most basic type of finite state automaton is the **finite state acceptor (FSA)**, which describes the computation involved in testing if a string is a member of a language. Formally, a finite state acceptor is a tuple $M = (Q, \Sigma, q_0, F, \delta)$, consisting of:

- a finite alphabet Σ of input symbols;
- a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- a start state $q_0 \in Q$;
- a set of final states $F \subseteq Q$;
- a transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$. The transition function maps from a state and an input symbol (or empty string ϵ) to a *set* of possible resulting states.

A **path** in M is a sequence of transitions, $\pi = t_1, t_2, \dots, t_N$, where each t_i traverses an arc in the transition function δ . The finite state acceptor M accepts a string ω if there is an accepting path, in which the initial transition t_1 begins at the start state q_0 , the final transition t_N terminates in a final state in Q , and the entire input ω is consumed.

Example

Consider the following FSA, M_1 .

$$\Sigma = \{a, b\} \quad [9.1]$$

$$Q = \{q_0, q_1\} \quad [9.2]$$

$$F = \{q_1\} \quad [9.3]$$

$$\delta = \{(q_0, a) \rightarrow q_0, (q_0, b) \rightarrow q_1, (q_1, b) \rightarrow q_1\}. \quad [9.4]$$

This FSA defines a language over an alphabet of two symbols, a and b . The transition function δ is written as a set of arcs: $(q_0, a) \rightarrow q_0$ says that if the machine is in state

4598 q_0 and reads symbol a , it stays in q_0 . Figure 9.1 provides a graphical representation of
 4599 M_1 . Because each pair of initial state and symbol has at most one resulting state, M_1 is
 4600 **deterministic**: each string ω induces at most one accepting path. Note that there are no
 4601 transitions for the symbol a in state q_1 ; if a is encountered in q_1 , then the acceptor is stuck,
 4602 and the input string is rejected.

4603 What strings does M_1 accept? The start state is q_0 , and we have to get to q_1 , since this
 4604 is the only final state. Any number of a symbols can be consumed in q_0 , but a b symbol is
 4605 required to transition to q_1 . Once there, any number of b symbols can be consumed, but
 4606 an a symbol cannot. So the regular expression corresponding to the language defined by
 4607 M_1 is a^*bb^* .

4608 Computational properties of finite state acceptors

4609 The key computational question for finite state acceptors is: how fast can we determine
 4610 whether a string is accepted? For deterministic FSAs, this computation can be performed
 4611 by Dijkstra's algorithm, with time complexity $\mathcal{O}(V \log V + E)$, where V is the number of
 4612 vertices in the FSA, and E is the number of edges (Cormen et al., 2009). Non-deterministic
 4613 FSAs (NFSAs) can include multiple transitions from a given symbol and state. Any NSFA
 4614 can be converted into a deterministic FSA, but the resulting automaton may have a num-
 4615 ber of states that is exponential in the number of size of the original NFSFA (Mohri et al.,
 4616 2002).

4617 9.1.2 Morphology as a regular language

4618 Many words have internal structure, such as prefixes and suffixes that shape their mean-
 4619 ing. The study of word-internal structure is the domain of **morphology**, of which there
 4620 are two main types:

- 4621 • **Derivational morphology** describes the use of affixes to convert a word from one
 4622 grammatical category to another (e.g., from the noun *grace* to the adjective *graceful*),
 4623 or to change the meaning of the word (e.g., from *grace* to *disgrace*).
- 4624 • **Inflectional morphology** describes the addition of details such as gender, number,
 4625 person, and tense (e.g., the *-ed* suffix for past tense in English).

4626 Morphology is a rich topic in linguistics, deserving of a course in its own right.¹ The
 4627 focus here will be on the use of finite state automata for morphological analysis. The

¹A good starting point would be a chapter from a linguistics textbook (e.g., Akmajian et al., 2010; Bender, 2013). A key simplification in this chapter is the focus on affixes at the sole method of derivation and inflection. English makes use of affixes, but also incorporates **apophony**, such as the inflection of *foot* to *feet*. Semitic languages like Arabic and Hebrew feature a template-based system of morphology, in which roots are triples of consonants (e.g., *ktb*), and words are created by adding vowels: *kataba* (Arabic: he wrote), *kutub* (books), *maktab* (desk). For more detail on morphology, see texts from Haspelmath and Sims (2013) and Lieber (2015).

4628 current section deals with derivational morphology; inflectional morphology is discussed
 4629 in § 9.1.4.

4630 Suppose that we want to write a program that accepts only those words that are con-
 4631 structed in accordance with the rules of English derivational morphology:

- 4632 (9.1) a. grace, graceful, gracefully, *gracelyful
 4633 b. disgrace, *ungrace, disgraceful, disgracefully
 4634 c. allure, *allureful, alluring, alluringly
 4635 d. fairness, unfair, *disfair, fairly

4636 (Recall that the asterisk indicates that a linguistic example is judged unacceptable by flu-
 4637 ent speakers of a language.) These examples cover only a tiny corner of English deriva-
 4638 tional morphology, but a number of things stand out. The suffix *-ful* converts the nouns
 4639 *grace* and *disgrace* into adjectives, and the suffix *-ly* converts adjectives into adverbs. These
 4640 suffixes must be applied in the correct order, as shown by the unacceptability of **grace-
 4641 lyful*. The *-ful* suffix works for only some words, as shown by the use of *alluring* as the
 4642 adjectival form of *allure*. Other changes are made with prefixes, such as the derivation
 4643 of *disgrace* from *grace*, which roughly corresponds to a negation; however, *fair* is negated
 4644 with the *un-* prefix instead. Finally, while the first three examples suggest that the direc-
 4645 tion of derivation is noun → adjective → adverb, the example of *fair* suggests that the
 4646 adjective can also be the base form, with the *-ness* suffix performing the conversion to a
 4647 noun.

4648 Can we build a computer program that accepts only well-formed English words, and
 4649 rejects all others? This might at first seem trivial to solve with a brute-force attack: simply
 4650 make a dictionary of all valid English words. But such an approach fails to account for
 4651 morphological **productivity** — the applicability of existing morphological rules to new
 4652 words and names, such as *Trump* to *Trumpy* and *Trumpkin*, and *Clinton* to *Clintonian* and
 4653 *Clintonite*. We need an approach that represents morphological rules explicitly, and for
 4654 this we will try a finite state acceptor.

4655 The dictionary approach can be implemented as a finite state acceptor, with the vo-
 4656 cabulary Σ equal to the vocabulary of English, and a transition from the start state to the
 4657 accepting state for each word. But this would of course fail to generalize beyond the origi-
 4658 nal vocabulary, and would not capture anything about the **morphotactic** rules that govern
 4659 derivations from new words. The first step towards a more general approach is shown in
 4660 Figure 9.2, which is the state diagram for a finite state acceptor in which the vocabulary
 4661 consists of **morphemes**, which include **stems** (e.g., *grace*, *allure*) and **affixes** (e.g., *dis-*, *-ing*,
 4662 *-ly*). This finite state acceptor consists of a set of paths leading away from the start state,
 4663 with derivational affixes added along the path. Except for q_{neg} , the states on these paths
 4664 are all final, so the FSA will accept *disgrace*, *disgraceful*, and *disgracefully*, but not *dis-*.

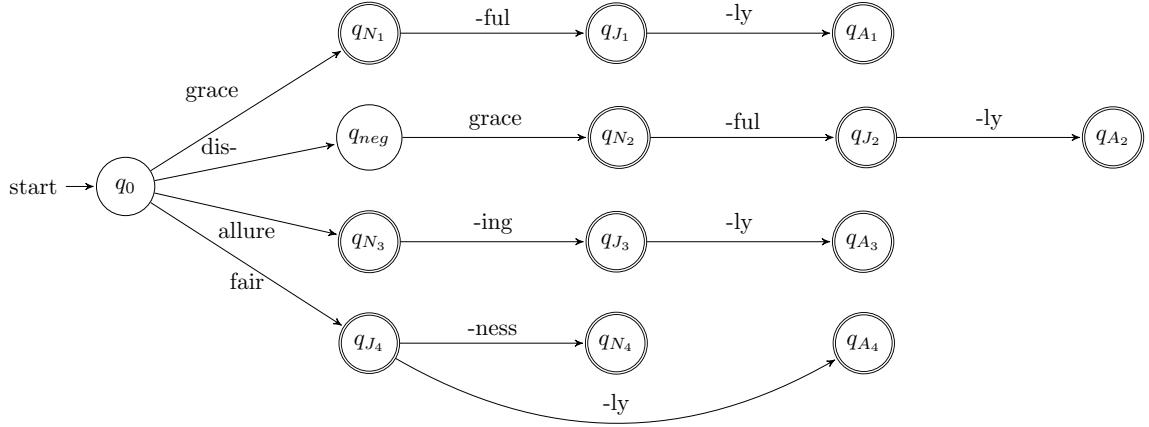


Figure 9.2: A finite state acceptor for a fragment of English derivational morphology. Each path represents possible derivations from a single root form.

4665 This FSA can be **minimized** to the form shown in Figure 9.3, which makes the general-
 4666 ity of the finite state approach more apparent. For example, the transition from q_0 to
 4667 q_{J_2} can be made to accept not only *fair* but any single-morpheme (**monomorphemic**) ad-
 4668 jective that takes *-ness* and *-ly* as suffixes. In this way, the finite state acceptor can easily
 4669 be extended: as new word stems are added to the vocabulary, their derived forms will be
 4670 accepted automatically. Of course, this FSA would still need to be extended considerably
 4671 to cover even this small fragment of English morphology. As shown by cases like *music*
 4672 \rightarrow *musical*, *athlete* \rightarrow *athletic*, English includes several classes of nouns, each with its own
 4673 rules for derivation.

4674 The FSAs shown in Figure 9.2 and 9.3 accept *allureing*, not *alluring*. This reflects a dis-
 4675 tinction between morphology — the question of which morphemes to use, and in what
 4676 order — and **orthography** — the question of how the morphemes are rendered in written
 4677 language. Just as orthography requires dropping the *e* preceding the *-ing* suffix, **phonol-**
 4678 **ogy** imposes a related set of constraints on how words are rendered in speech. As we will
 4679 see soon, these issues can be handled by **finite state transducers**, which are finite state
 4680 automata that take inputs and produce outputs.

4681 9.1.3 Weighted finite state acceptors

4682 According to the FSA treatment of morphology, every word is either in or out of the lan-
 4683 guage, with no wiggle room. Perhaps you agree that *musicky* and *fishful* are not valid
 4684 English words; but if forced to choose, you probably find *a fishful stew* or *a musicky trib-*
 4685 *ute* preferable to *behaving disgracelyful*. Rather than asking whether a word is acceptable,
 4686 we might like to ask how acceptable it is. Aronoff (1976, page 36) puts it another way:

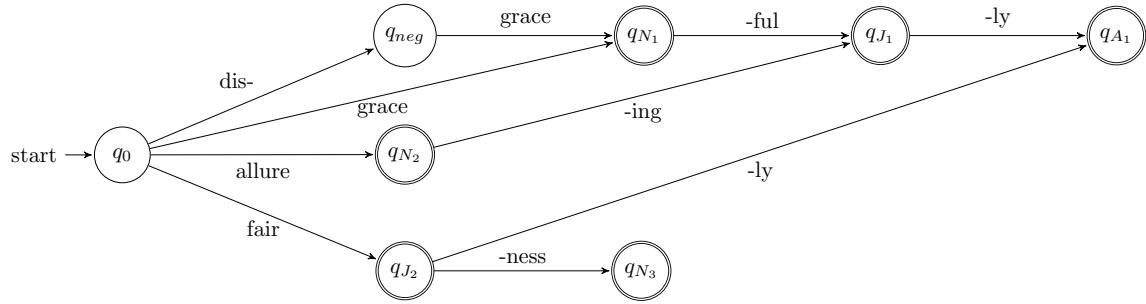


Figure 9.3: Minimization of the finite state acceptor shown in Figure 9.2.

4687 “Though many things are possible in morphology, some are more possible than others.”
 4688 But finite state acceptors give no way to express preferences among technically valid
 4689 choices.

4690 **Weighted finite state acceptors (WFSAs)** are generalizations of FSAs, in which each
 4691 accepting path is assigned a score, computed from the transitions, the initial state, and the
 4692 final state. Formally, a weighted finite state acceptor $M = (Q, \Sigma, \lambda, \rho, \delta)$ consists of:

- 4693 • a finite set of states $Q = \{q_0, q_1, \dots, q_n\}$;
- 4694 • a finite alphabet Σ of input symbols;
- 4695 • an initial weight function, $\lambda : Q \rightarrow \mathbb{R}$;
- 4696 • a final weight function $\rho : Q \rightarrow \mathbb{R}$;
- 4697 • a transition function $\delta : Q \times \Sigma \times Q \rightarrow \mathbb{R}$.

4698 WFSAs depart from the FSA formalism in three ways: every state can be an initial
 4699 state, with score $\lambda(q)$; every state can be an accepting state, with score $\rho(q)$; transitions are
 4700 possible between any pair of states on any input, with a score $\delta(q_i, \omega, q_j)$. Nonetheless,
 4701 FSAs can be viewed as a special case: for any FSA M we can build an equivalent WPSA
 4702 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
 4703 transitions $\{(q_1, \omega) \rightarrow q_2\}$ that are not permitted by the transition function of M .

4704 The total score for any path $\pi = t_1, t_2, \dots, t_N$ is equal to the sum of these scores,

$$d(\pi) = \lambda(\text{from-state}(t_1)) + \sum_n^N \delta(t_n) + \rho(\text{to-state}(t_N)). \quad [9.5]$$

4705 A **shortest-path algorithm** is used to find the minimum-cost path through a WPSA for
 4706 string ω , with time complexity $\mathcal{O}(E + V \log V)$, where E is the number of edges and V is
 4707 the number of vertices (Cormen et al., 2009).²

²Shortest-path algorithms find the path with the minimum cost. In many cases, the path weights are log

4708 **N-gram language models as WFSAs**

4709 In **n-gram language models** (see § 6.1), the probability of a sequence of tokens w_1, w_2, \dots, w_M
 4710 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:

$$\begin{aligned} \delta(q_i, \omega, q_j) &= \begin{cases} \log \Pr(w_m = j | w_{m-1} = i), & \omega = j \\ -\infty, & \omega \neq j \end{cases} \\ \lambda(q_i) &= \log \Pr(w_1 = i | w_0 = \square) \\ \rho(q_i) &= \log \Pr(w_{M+1} = \blacksquare | w_M = i). \end{aligned}$$

4711 The transition function is designed to ensure that the context is recorded accurately:
 4712 we can move to state j on input ω only if $\omega = j$; otherwise, transitioning to state j is
 4713 forbidden by the weight of $-\infty$. The initial weight function $\lambda(q_i)$ is the log probability of
 4714 receiving i as the first token, and the final weight function $\rho(q_i)$ is the log probability of
 4715 receiving an “end-of-string” token after observing $w_M = i$.

4716 ***Semiring weighted finite state acceptors**

4717 The n -gram language model WFSA is deterministic: each input has exactly one accepting
 4718 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.

4719 may have multiple accepting paths. In some applications, the score for the input is ag-
 4720 gregated across all such paths. Such aggregate scores can be computed by generalizing
 4721 WFSAs with **semiring notation**, first introduced in § 7.7.3.

4722 Let $d(\pi)$ represent the total score for path $\pi = t_1, t_2, \dots, t_N$, which is computed as,

$$d(\pi) = \lambda(\text{from-state}(t_1)) \otimes \delta(t_1) \otimes \delta(t_2) \otimes \dots \otimes \delta(t_N) \otimes \rho(\text{to-state}(t_N)). \quad [9.8]$$

4723 This is a generalization of Equation 9.5 to semiring notation, using the semiring multipli-
 4724 cation operator \otimes in place of addition.

4725 Now let $s(\omega)$ represent the total score for all paths $\Pi(\omega)$ that consume input ω ,

$$s(\omega) = \bigoplus_{\pi \in \Pi(\omega)} d(\pi). \quad [9.9]$$

4726 Here, semiring addition (\oplus) is used to combine the scores of multiple paths.

4727 The generalization to semirings covers a number of useful special cases. In the log-
 4728 probability semiring, multiplication is defined as $\log p(x) \otimes \log p(y) = \log p(x) + \log p(y)$,
 4729 and addition is defined as $\log p(x) \oplus \log p(y) = \log(p(x) + p(y))$. Thus, $s(\omega)$ represents
 4730 the log-probability of accepting input ω , marginalizing over all paths $\pi \in \Pi(\omega)$. In the
 4731 **boolean semiring**, the \otimes operator is logical conjunction, and the \oplus operator is logical
 4732 disjunction. This reduces to the special case of unweighted finite state acceptors, where
 4733 the score $s(\omega)$ is a boolean indicating whether there exists any accepting path for ω . In
 4734 the **tropical semiring**, the \oplus operator is a maximum, so the resulting score is the score of
 4735 the best-scoring path through the WFSAs. The OPENFST toolkit uses semirings and poly-
 4736 morphism to implement general algorithms for weighted finite state automata (Allauzen
 4737 et al., 2007).

4738 *Interpolated n -gram language models

4739 Recall from § 6.2.3 that an **interpolated n -gram language model** combines the probabili-
 4740 ties from multiple n -gram models. For example, an interpolated bigram language model
 4741 computes the probability,

$$\hat{p}(w_m | w_{m-1}) = \lambda_1 p_1(w_m) + \lambda_2 p_2(w_m | w_{m-1}), \quad [9.10]$$

4742 with \hat{p} indicating the interpolated probability, p_2 indicating the bigram probability, and
 4743 p_1 indicating the unigram probability. Setting $\lambda_2 = (1 - \lambda_1)$ ensures that the probabilities
 4744 sum to one.

4745 Interpolated bigram language models can be implemented using a non-deterministic
 4746 WFSAs (Knight and May, 2009). The basic idea is shown in Figure 9.4. In an interpolated
 4747 bigram language model, there is one state for each element in the vocabulary — in this

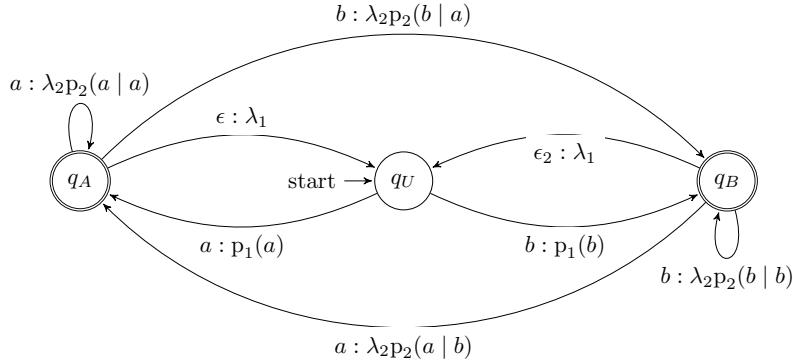


Figure 9.4: WFSA implementing an interpolated bigram/unigram language model, on the alphabet $\Sigma = \{a, b\}$. For simplicity, the WFSA is constrained to force the first token to be generated from the unigram model, and does not model the emission of the end-of-sequence token.

4748 case, the states q_A and q_B — which capture the contextual conditioning in the bigram
 4749 probabilities. To model unigram probabilities, there is an additional state q_U , which “for-
 4750 gets” the context. Transitions out of q_U involve unigram probabilities, $p_1(a)$ and $p_2(b)$;
 4751 transitions into q_U emit the empty symbol ϵ , and have probability λ_1 , reflecting the inter-
 4752 polation weight for the unigram model. The interpolation weight for the bigram model is
 4753 included in the weight of the transition $q_A \rightarrow q_B$.

4754 The epsilon transitions into q_U make this WFSA non-deterministic. Consider the score
 4755 for the sequence (a, b, b) . The initial state is q_U , so the symbol a is generated with score
 4756 $p_1(a)$ ³ Next, we can generate b from the unigram model by taking the transition $q_A \rightarrow q_B$,
 4757 with score $\lambda_2 p_2(b | a)$. Alternatively, we can take a transition back to q_U with score λ_1 ,
 4758 and then emit b from the unigram model with score $p_1(b)$. To generate the final b token,
 4759 we face the same choice: emit it directly from the self-transition to q_B , or transition to q_U
 4760 first.

The total score for the sequence (a, b, b) is the semiring sum over all accepting paths,

$$\begin{aligned}
 s(a, b, b) &= (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes \lambda_2 p_2(b | b)) \\
 &\oplus (p_1(a) \otimes \lambda_2 p_2(b | a) \otimes p_1(b) \otimes p_1(b)) \\
 &\oplus (p_1(a) \otimes \lambda_1 \otimes p_1(b) \otimes p_1(b) \otimes p_1(b)). \tag{[9.11]}
 \end{aligned}$$

4761 Each line in Equation 9.11 represents the probability of a specific path through the WFSA.
 4762 In the probability semiring, \otimes is multiplication, so that each path is the product of each

³We could model the sequence-initial bigram probability $p_2(a | \square)$, but for simplicity the WFSA does not admit this possibility, which would require another state.

4763 transition weight, which are themselves probabilities. The \oplus operator is addition, so that
 4764 the total score is the sum of the scores (probabilities) for each path. This corresponds to
 4765 the probability under the interpolated bigram language model.

4766 **9.1.4 Finite state transducers**

4767 Finite state acceptors can determine whether a string is in a regular language, and weighted
 4768 finite state acceptors can compute a score for every string over a given alphabet. **Finite**
 4769 **state transducers** (FSTs) extend the formalism further, by adding an output symbol to each
 4770 transition. Formally, a finite state transducer is a tuple $T = (Q, \Sigma, \Omega, \lambda, \rho, \delta)$, with Ω repre-
 4771 senting an output vocabulary and the transition function $\delta : Q \times (\Sigma \cup \epsilon) \times (\Omega \cup \epsilon) \times Q \rightarrow \mathbb{R}$
 4772 mapping from states, input symbols, and output symbols to states. The remaining ele-
 4773 ments (Q, Σ, λ, ρ) are identical to their definition in weighted finite state acceptors (§ 9.1.3).
 4774 Thus, each path through the FST T transduces the input string into an output.

4775 **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,

$$\delta(q, a, a, q) = \delta(q, b, b, q) = 0 \quad [9.12]$$

$$\delta(q, a, b, q) = \delta(q, b, a, q) = 1 \quad [9.13]$$

$$\delta(q, a, \epsilon, q) = \delta(q, b, \epsilon, q) = 1 \quad [9.14]$$

$$\delta(q, \epsilon, a, q) = \delta(q, \epsilon, b, q) = 1. \quad [9.15]$$

4776 The state diagram is shown in Figure 9.5.

4777 For a given string pair, there are multiple paths through the transducer: the best-
 4778 scoring path from *dessert* to *desert* involves a single deletion, for a total score of 1; the
 4779 worst-scoring path involves seven deletions and six additions, for a score of 13.

4780 **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

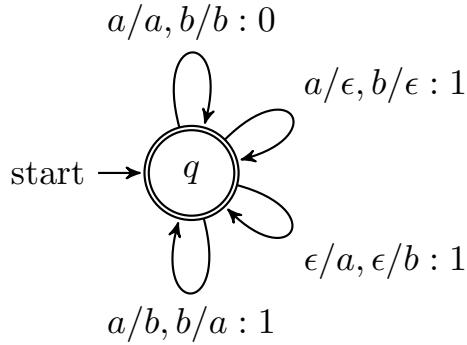


Figure 9.5: State diagram for the Levenshtein edit distance finite state transducer. The label $x/y : c$ indicates a cost of c for a transition with input x and output y .

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]

4781 The final two lines appear to conflict; they are meant to be interpreted as an instruction
 4782 to remove a terminal *-s* unless it is part of an *-ss* ending. A state diagram to handle just
 4783 these final two lines is shown in Figure 9.6. Make sure you understand how this finite
 4784 state transducer handles *cats*, *steps*, *bass*, and *basses*.

4785 Inflectional morphology

4786 In **inflectional morphology**, word **lemmas** are modified to add grammatical information
 4787 such as tense, number, and case. For example, many English nouns are pluralized by the
 4788 suffix *-s*, and many verbs are converted to past tense by the suffix *-ed*. English's inflectional
 4789 morphology is considerably simpler than many of the world's languages. For example,
 4790 Romance languages (derived from Latin) feature complex systems of verb suffixes which
 4791 must agree with the person and number of the verb, as shown in Table 9.1.

4792 The task of morphological analysis is to read a form like *canto*, and output an analysis
 4793 like CANTAR+VERB+PRESIND+1P+SING, where +PRESIND describes the tense as present
 4794 indicative, +1P indicates the first-person, and +SING indicates the singular number. The
 4795 task of morphological generation is the reverse, going from CANTAR+VERB+PRESIND+1P+SING
 4796 to *canto*. Finite state transducers are an attractive solution, because they can solve both
 4797 problems with a single model (Beesley and Karttunen, 2003). As an example, Figure 9.7
 4798 shows a fragment of a finite state transducer for Spanish inflectional morphology. The

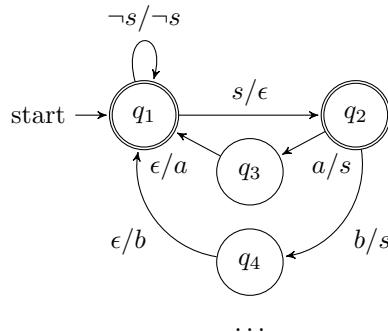


Figure 9.6: State diagram for final two lines of step 1a of the Porter stemming diagram. States q_3 and q_4 “remember” the observations a and b respectively; the ellipsis \dots represents additional states for each symbol in the input alphabet. The notation $\neg s / \neg s$ is not part of the FST formalism; 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.

4799 input vocabulary Σ corresponds to the set of letters used in Spanish spelling, and the out-
 4800 put vocabulary Ω corresponds to these same letters, plus the vocabulary of morphological
 4801 features (e.g., +SING, +VERB). In Figure 9.7, there are two paths that take *canto* as input,
 4802 corresponding to the verb and noun meanings; the choice between these paths could be
 4803 guided by a part-of-speech tagger. By **inversion**, the inputs and outputs for each trans-
 4804 sition are switched, resulting in a finite state generator, capable of producing the correct
 4805 **surface form** for any morphological analysis.

4806 Finite state morphological analyzers and other unweighted transducers can be de-
 4807 signed by hand. The designer’s goal is to avoid **overgeneration** — accepting strings or
 4808 making transductions that are not valid in the language — as well as **undergeneration**

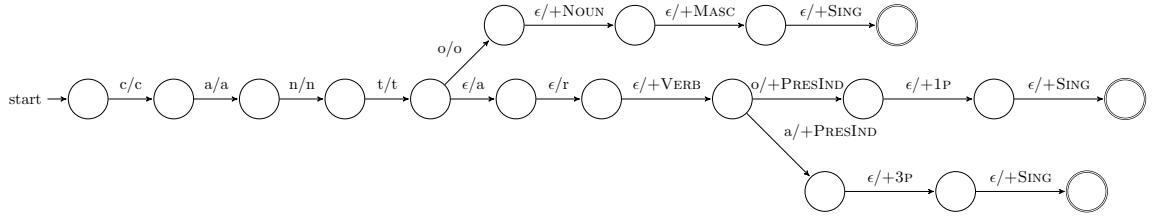


Figure 9.7: Fragment of a finite state transducer for Spanish morphology. There are two accepting paths for the input *canto*: *canto+NOUN+MASC+SING* (masculine singular noun, meaning a song), and *cantar+VERB+PRESIND+1P+SING* (I sing). There is also an accepting path for *canta*, with output *cantar+VERB+PRESIND+3P+SING* (he/she sings).

4809 — failing to accept strings or transductions that are valid. For example, a pluralization
 4810 transducer that does not accept *foot/feet* would undergenerate. Suppose we “fix” the trans-
 4811 ducer to accept this example, but as a side effect, it now accepts *boot/beet*; the transducer
 4812 would then be said to overgenerate. If a transducer accepts *foot/foots* but not *foot/feet*, then
 4813 it simultaneously overgenerates and undergenerates.

4814 Finite state composition

4815 Designing finite state transducers to capture the full range of morphological phenomena
 4816 in any real language is a huge task. Modularization is a classic computer science approach
 4817 for this situation: decompose a large and unwieldy problem into a set of subproblems,
 4818 each of which will hopefully have a concise solution. Finite state automata can be mod-
 4819 ularized through **composition**: feeding the output of one transducer T_1 as the input to
 4820 another transducer T_2 , written $T_2 \circ T_1$. Formally, if there exists some y such that $(x, y) \in T_1$
 4821 (meaning that T_1 produces output y on input x), and $(y, z) \in T_2$, then $(x, z) \in (T_2 \circ T_1)$.
 4822 Because finite state transducers are closed under composition, there is guaranteed to be
 4823 a single finite state transducer that $T_3 = T_2 \circ T_1$, which can be constructed as a machine
 4824 with one state for each pair of states in T_1 and T_2 (Mohri et al., 2002).

4825 **Example: Morphology and orthography** In English morphology, the suffix *-ed* is added
 4826 to signal the past tense for many verbs: *cook*→*cooked*, *want*→*wanted*, etc. However, English
 4827 **orthography** dictates that this process cannot produce a spelling with consecutive e’s, so
 4828 that *bake*→*baked*, not *bakeed*. A modular solution is to build separate transducers for mor-
 4829 phology and orthography. The morphological transducer T_M transduces from *bake+PAST*
 4830 to *bake+ed*, with the + symbol indicating a segment boundary. The input alphabet of T_M
 4831 includes the lexicon of words and the set of morphological features; the output alphabet
 4832 includes the characters *a-z* and the + boundary marker. Next, an orthographic transducer
 4833 T_O is responsible for the transductions *cook+ed*→*cooked*, and *bake+ed*→*baked*. The input
 4834 alphabet of T_O must be the same as the output alphabet for T_M , and the output alphabet

4835 is simply the characters *a-z*. The composed transducer ($T_O \circ T_M$) then transduces from
 4836 *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,

$$\delta(q_{m-1,i}, Y_m = j, q_{m,j}) = \log p(w_m, Y_m = j \mid Y_{m-1} = i) \quad [9.20]$$

$$= \log p(w_m \mid Y_m = j) + \log \Pr(Y_m = j \mid Y_{m-1} = i). \quad [9.21]$$

4837 Because there is only one possible transition for each tag Y_m , this WFSA is deterministic.
 4838 The score for any tag sequence $\{y_m\}_{m=1}^M$ is the sum of these log-probabilities, correspond-
 4839 ing to the total log probability $\log p(w, y)$. Furthermore, the trellis can be constructed by
 4840 the composition of simpler FSTs.

- 4841 • First, construct a “transition” transducer to represent a bigram probability model
 4842 over tag sequences, T_T . This transducer is almost identical to the n -gram language
 4843 model acceptor in § 9.1.3: there is one state for each tag, and the edge weights equal
 4844 to the transition log-probabilities, $\delta(q_i, j, j, q_j) = \log \Pr(Y_m = j \mid Y_{m-1} = i)$. Note
 4845 that T_T is a transducer, with identical input and output at each arc; this makes it
 4846 possible to compose T_T with other transducers.
- 4847 • Next, construct an “emission” transducer to represent the probability of words given
 4848 tags, T_E . This transducer has only a single state, with arcs for each word/tag pair,
 4849 $\delta(q_0, i, j, q_0) = \log \Pr(W_m = j \mid Y_m = i)$. The input vocabulary is the set of all tags,
 4850 and the output vocabulary is the set of all words.
- 4851 • The composition $T_E \circ T_T$ is a finite state transducer with one state per tag, as shown
 4852 in Figure 9.8. Each state has $V \times K$ outgoing edges, representing transitions to each
 4853 of the K other states, with outputs for each of the V words in the vocabulary. The
 4854 weights for these edges are equal to,

$$\delta(q_i, Y_m = j, w_m, q_j) = \log p(w_m, Y_m = j \mid Y_{m-1} = i). \quad [9.22]$$

- 4855 • The trellis is a structure with $M \times K$ nodes, for each of the M words to be tagged and
 4856 each of the K tags in the tagset. It can be built by composition of $(T_E \circ T_T)$ against an
 4857 unweighted **chain FSA** $M_A(w)$ that is specially constructed to accept only a given
 4858 input w_1, w_2, \dots, w_M , shown in Figure 9.9. The trellis for input w is built from the
 4859 composition $M_A(w) \circ (T_E \circ T_T)$. Composing with the unweighted $M_A(w)$ does not
 4860 affect the edge weights from $(T_E \circ T_T)$, but it selects the subset of paths that generate
 4861 the word sequence w .

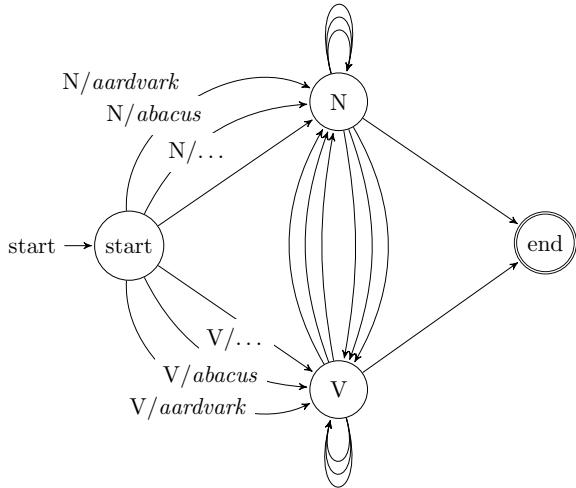


Figure 9.8: Finite state transducer for hidden Markov models, with a small tagset of nouns and verbs. For each pair of tags (including self-loops), there is an edge for every word in the vocabulary. For simplicity, input and output are only shown for the edges from the start state. Weights are also omitted 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 $\log \Pr(Y_m = \diamond \mid Y_{m-1} = i)$.

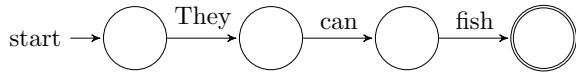


Figure 9.9: Chain finite state acceptor for the input *They can fish*.

4862 9.1.5 *Learning weighted finite state automata

4863 In generative models such as n -gram language models and hidden Markov models, the
 4864 edge weights correspond to log probabilities, which can be obtained from relative fre-
 4865 quency estimation. However, in other cases, we wish to learn the edge weights from in-
 4866 put/output pairs. This is difficult in non-deterministic finite state automata, because we
 4867 do not observe the specific arcs that are traversed in accepting the input, or in transducing
 4868 from input to output. The path through the automaton is a **latent variable**.

4869 Chapter 5 presented one method for learning with latent variables: expectation max-
 4870 imization (EM). This involves computing a distribution $q(\cdot)$ over the latent variable, and
 4871 iterating between updates to this distribution and updates to the parameters — in this
 4872 case, the arc weights. The **forward-backward algorithm** (§ 7.5.3) describes a dynamic
 4873 program for computing a distribution over arcs in the trellis structure of a hidden Markov

model, but this is a special case of the more general problem for finite state automata. Eisner (2002) describes an **expectation semiring**, which enables the expected number of transitions across each arc to be computed through a semiring shortest-path algorithm. Alternative approaches for generative models include Markov Chain Monte Carlo (Chiang et al., 2010) and spectral learning (Balle et al., 2011).

Further afield, we can take a perceptron-style approach, with each arc corresponding to a feature. The classic perceptron update would update the weights by subtracting the difference between the feature vector corresponding to the predicted path and the feature vector corresponding to the correct path. Since the path is not observed, we resort to a **latent variable perceptron**. The model is described formally in § 12.4, but the basic idea is to compute an update from the difference between the features from the predicted path and the features for the best-scoring path that generates the correct output.

9.2 Context-free languages

Beyond the class of regular languages lie the context-free languages. An example of a language that is context-free but not finite state is the set of arithmetic expressions with balanced parentheses. Intuitively, to accept only strings in this language, an FSA would have to “count” the number of left parentheses, and make sure that they are balanced against the number of right parentheses. An arithmetic expression can be arbitrarily long, yet by definition an FSA has a finite number of states. Thus, for any FSA, there will be a string that with too many parentheses to count. More formally, the **pumping lemma** is a proof technique for showing that languages are not regular. It is typically demonstrated for the simpler case $a^n b^n$, the language of strings containing a sequence of a 's, and then an equal-length sequence of b 's.⁴

There are at least two arguments for the relevance of non-regular formal languages to linguistics. First, there are natural language phenomena that are argued to be isomorphic to $a^n b^n$. For English, the classic example is **center embedding**, shown in Figure 9.10. The initial expression *the dog* specifies a single dog. Embedding this expression into *the cat ... chased* specifies a particular cat — the one chased by the dog. This cat can then be embedded again to specify a goat, in the less felicitous but arguably grammatical expression, *the goat the cat the dog chased kissed*, which refers to the goat who was kissed by the cat which was chased by the dog. Chomsky (1957) argues that to be grammatical, a center-embedded construction must be balanced: if it contains n noun phrases (e.g., *the cat*), they must be followed by exactly $n - 1$ verbs. An FSA that could recognize such expressions would also be capable of recognizing the language $a^n b^n$. Because we can prove that no FSA exists for $a^n b^n$, no FSA can exist for center embedded constructions either. En-

⁴Details of the proof can be found in an introductory computer science theory textbook (e.g., Sipser, 2012).

| | | | | |
|----------|---------|---------|---------|--------|
| | | | the dog | |
| | the cat | the dog | chased | |
| the goat | the cat | the dog | chased | kissed |
| | | | ... | |

Figure 9.10: Three levels of center embedding

4909 glish includes center embedding, and so the argument goes, English grammar as a whole
 4910 cannot be regular.⁵

4911 A more practical argument for moving beyond regular languages is modularity. Many
 4912 linguistic phenomena — especially in syntax — involve constraints that apply at long
 4913 distance. Consider the problem of determiner-noun number agreement in English: we
 4914 can say *the coffee* and *these coffees*, but not **these coffee*. By itself, this is easy enough to model
 4915 in an FSA. However, fairly complex modifying expressions can be inserted between the
 4916 determiner and the noun:

- 4917 (9.2) a. the burnt coffee
 4918 b. the badly-ground coffee
 4919 c. the burnt and badly-ground Italian coffee
 4920 d. these burnt and badly-ground Italian coffees
 4921 e. * these burnt and badly-ground Italian coffee

4922 Again, an FSA can be designed to accept modifying expressions such as *burnt and badly-*
 4923 *ground Italian*. Let's call this FSA F_M . To reject the final example, a finite state acceptor
 4924 must somehow "remember" that the determiner was plural when it reaches the noun *cof-*
 4925 *fee* at the end of the expression. The only way to do this is to make two identical copies
 4926 of F_M : one for singular determiners, and one for plurals. While this is possible in the
 4927 finite state framework, it is inconvenient — especially in languages where more than one
 4928 attribute of the noun is marked by the determiner. **Context-free languages** facilitate mod-
 4929 ularity across such long-range dependencies.

4930 9.2.1 Context-free grammars

4931 Context-free languages are specified by **context-free grammars** (CFGs), which are tuples
 4932 (N, Σ, R, S) consisting of:

⁵The claim that arbitrarily deep center-embedded expressions are grammatical has drawn skepticism. Corpus evidence shows that embeddings of depth greater than two are exceedingly rare (Karlsson, 2007), and that embeddings of depth greater than three are completely unattested. If center-embedding is capped at some finite depth, then it is regular.

$$\begin{aligned}
 S &\rightarrow S \text{ OP } S \mid \text{NUM} \\
 \text{OP} &\rightarrow + \mid - \mid \times \mid \div \\
 \text{NUM} &\rightarrow \text{NUM DIGIT} \mid \text{DIGIT} \\
 \text{DIGIT} &\rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9
 \end{aligned}$$

Figure 9.11: A context-free grammar for arithmetic expressions

- 4933 • a finite set of **non-terminals** N ;
- 4934 • a finite alphabet Σ of **terminal symbols**;
- 4935 • a set of **production rules** R , each of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$;
- 4936 • a designated start symbol S .

4937 In the production rule $A \rightarrow \beta$, the left-hand side (LHS) A must be a non-terminal;
 4938 the right-hand side (RHS) can be a sequence of terminals or non-terminals, $\{n, \sigma\}^*, n \in$
 4939 $N, \sigma \in \Sigma$. A non-terminal can appear on the left-hand side of many production rules.
 4940 A non-terminal can appear on both the left-hand side and the right-hand side; this is a
 4941 **recursive production**, and is analogous to self-loops in finite state automata. The name
 4942 “context-free” is based on the property that the production rule depends only on the LHS,
 4943 and not on its ancestors or neighbors; this is analogous to Markov property of finite state
 4944 automata, in which the behavior at each step depends only on the current state, on not on
 4945 the path by which that state was reached.

4946 A **derivation** τ is a sequence of steps from the start symbol S to a surface string $w \in \Sigma^*$,
 4947 which is the **yield** of the derivation. A string w is in a context-free language if there is
 4948 some derivation from S yielding w . **Parsing** is the problem of finding a derivation for a
 4949 string in a grammar. Algorithms for parsing are described in chapter 10.

4950 Like regular expressions, context-free grammars define the language but not the com-
 4951 putation necessary to recognize it. The context-free analogues to finite state acceptors are
 4952 **pushdown automata**, a theoretical model of computation in which input symbols can be
 4953 pushed onto a stack with potentially infinite depth. For more details, see Sipser (2012).

4954 Example

4955 Figure 9.11 shows a context-free grammar for arithmetic expressions such as $1 + 2 \div 3 - 4$.
 4956 In this grammar, the terminal symbols include the digits $\{1, 2, \dots, 9\}$ and the op-
 4957 erators $\{+, -, \times, \div\}$. The rules include the $|$ symbol, a notational convenience that makes
 4958 it possible to specify multiple right-hand sides on a single line: the statement $A \rightarrow x | y$

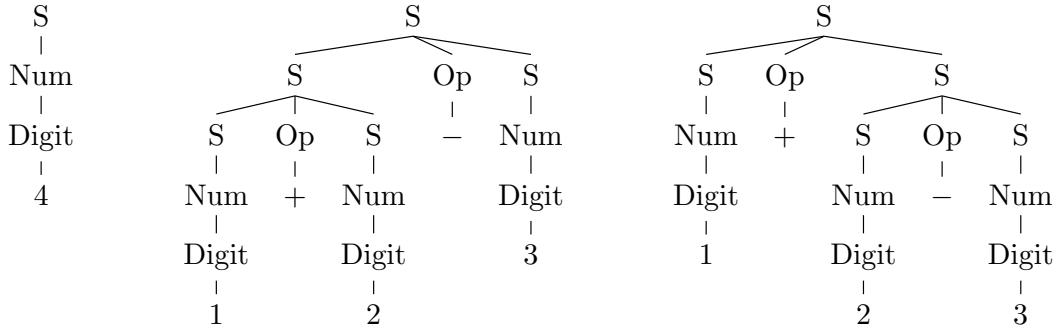


Figure 9.12: Some example derivations from the arithmetic grammar in Figure 9.11

4959 defines *two* productions, $A \rightarrow x$ and $A \rightarrow y$. This grammar is recursive: the non-termals S
4960 and NUM can produce themselves.

4961 Derivations are typically shown as trees, with production rules applied from the top
4962 to the bottom. The tree on the left in Figure 9.12 describes the derivation of a single digit,
4963 through the sequence of productions $S \rightarrow \text{NUM} \rightarrow \text{DIGIT} \rightarrow 4$ (these are all **unary produc-**
4964 **tions**, because the right-hand side contains a single element). The other two trees in
4965 Figure 9.12 show alternative derivations of the string $1 + 2 - 3$. The existence of multiple
4966 derivations for a string indicates that the grammar is **ambiguous**.

Context-free derivations can also be written out according to the pre-order tree traversal.⁶ For the two derivations of $1 + 2 - 3$ in Figure 9.12, the notation is:

$$(S (S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2))))) (\text{Op} -) (S (\text{Num} (Digit 3)))) \quad [9.23]$$

$$(S (S (\text{Num} (Digit 1))) (\text{Op} +) (S (\text{Num} (Digit 2)) (\text{Op} -) (S (\text{Num} (Digit 3)))))). \quad [9.24]$$

4967 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 $n > 0$.

$$\begin{aligned} S &\rightarrow aSb \mid ab \\ S &\rightarrow aSb \mid aabb \mid ab \end{aligned}$$

4968 Two grammars are **weakly equivalent** if they generate the same strings. Two grammars
4969 are **strongly equivalent** if they generate the same strings via the same derivations. The
4970 grammars above are only weakly equivalent.

⁶This is a depth-first left-to-right search that prints each node the first time it is encountered (Cormen et al., 2009, chapter 12).

In **Chomsky Normal Form (CNF)**, the right-hand side of every production includes either two non-terminals, or a single terminal symbol:

$$A \rightarrow BC$$

$$A \rightarrow a$$

- 4971 All CFGs can be converted into a CNF grammar that is weakly equivalent. To convert a
 4972 grammar into CNF, we first address productions that have more than two non-terminals
 4973 on the RHS by creating new “dummy” non-terminals. For example, if we have the pro-
 4974 duction,

$$W \rightarrow X Y Z, \quad [9.25]$$

it is replaced with two productions,

$$W \rightarrow X W \setminus X \quad [9.26]$$

$$W \setminus X \rightarrow Y Z. \quad [9.27]$$

- 4975 In these productions, $W \setminus X$ is a new dummy non-terminal. This transformation **binarizes**
 4976 the grammar, which is critical for efficient bottom-up parsing, as we will see in chapter 10.
 4977 Productions whose right-hand side contains a mix of terminal and non-terminal symbols
 4978 can be replaced in a similar fashion.

- 4979 Unary non-terminal productions $A \rightarrow B$ are replaced as follows: for each production
 4980 $B \rightarrow \alpha$ in the grammar, add a new production $A \rightarrow \alpha$. For example, in the grammar
 4981 described in Figure 9.11, we would replace $\text{NUM} \rightarrow \text{DIGIT}$ with $\text{NUM} \rightarrow 1 \mid 2 \mid \dots \mid 9$.
 4982 However, we keep the production $\text{NUM} \rightarrow \text{NUM DIGIT}$, which is a valid binary produc-
 4983 tion.

4984 9.2.2 Natural language syntax as a context-free language

- 4985 Context-free grammars can be used to represent **syntax**, which is the set of rules that
 4986 determine whether an utterance is judged to be grammatical. If this representation were
 4987 perfectly faithful, then a natural language such as English could be transformed into a
 4988 formal language, consisting of exactly the (infinite) set of strings that would be judged to
 4989 be grammatical by a fluent English speaker. We could then build parsing software that
 4990 would automatically determine if a given utterance were grammatical.⁷

- 4991 Contemporary theories generally do *not* consider natural languages to be context-free
 4992 (see § 9.3), yet context-free grammars are widely used in natural language parsing. The
 4993 reason is that context-free representations strike a good balance: they cover a broad range
 4994 of syntactic phenomena, and they can be parsed efficiently. This section therefore de-
 4995 scribes how to handle a core fragment of English syntax in context-free form, following

⁷To move beyond this cursory treatment of syntax, consult the short introductory manuscript by Bender (2013), or the longer text by Akmajian et al. (2010).

4996 the conventions of the **Penn Treebank** (PTB; Marcus et al., 1993), a large-scale annotation
 4997 of English language syntax. The generalization to “mildly” context-sensitive languages is
 4998 discussed in § 9.3.

4999 The Penn Treebank annotation is a **phrase-structure grammar** of English. This means
 5000 that sentences are broken down into **constituents**, which are contiguous sequences of
 5001 words that function as coherent units for the purpose of linguistic analysis. Constituents
 5002 generally have a few key properties:

5003 **Movement.** Constituents can often be moved around sentences as units.

- 5004 (9.3) a. Abigail gave (her brother) (a fish).
 5005 b. Abigail gave (a fish) to (her brother).

5006 In contrast, *gave her* and *brother a* cannot easily be moved while preserving gram-
 5007 maticality.

5008 **Substitution.** Constituents can be substituted by other phrases of the same type.

- 5009 (9.4) a. Max thanked (his older sister).
 5010 b. Max thanked (her).

5011 In contrast, substitution is not possible for other contiguous units like *Max thanked*
 5012 and *thanked his*.

5013 **Coordination.** Coordinators like *and* and *or* can conjoin constituents.

- 5014 (9.5) a. (Abigail) and (her younger brother) bought a fish.
 5015 b. Abigail (bought a fish) and (gave it to Max).
 5016 c. Abigail (bought) and (greedily ate) a fish.

5017 Units like *brother bought* and *bought a* cannot easily be coordinated.

5018 These examples argue for units such as *her brother* and *bought a fish* to be treated as con-
 5019 stituents. Other sequences of words in these examples, such as *Abigail gave* and *brother a fish*,
 5020 cannot be moved, substituted, and coordinated in these ways. In phrase-structure
 5021 grammar, constituents are nested, so that *the senator from New Jersey* contains the con-
 5022 stituent *from New Jersey*, which in turn contains *New Jersey*. The sentence itself is the max-
 5023 imal constituent; each word is a minimal constituent, derived from a unary production
 5024 from a part-of-speech tag. Between part-of-speech tags and sentences are **phrases**. In
 5025 phrase-structure grammar, phrases have a type that is usually determined by their **head**
 5026 **word**: for example, a **noun phrase** corresponds to a noun and the group of words that

5027 modify it, such as *her younger brother*; a **verb phrase** includes the verb and its modifiers,
5028 such as *bought a fish* and *greedily ate it*.

5029 In context-free grammars, each phrase type is a non-terminal, and each constituent is
5030 the substring that the non-terminal yields. Grammar design involves choosing the right
5031 set of non-terminals. Fine-grained non-terminals make it possible to represent more fine-
5032 grained linguistic phenomena. For example, by distinguishing singular and plural noun
5033 phrases, it is possible to have a grammar of English that generates only sentences that
5034 obey subject-verb agreement. However, enforcing subject-verb agreement is considerably
5035 more complicated in languages like Spanish, where the verb must agree in both person
5036 and number with subject. In general, grammar designers must trade off between **over-**
5037 **generation** — a grammar that permits ungrammatical sentences — and **undergeneration**
5038 — a grammar that fails to generate grammatical sentences. Furthermore, if the grammar is
5039 to support manual annotation of syntactic structure, it must be simple enough to annotate
5040 efficiently.

5041 9.2.3 A phrase-structure grammar for English

5042 To better understand how phrase-structure grammar works, let's consider the specific
5043 case of the Penn Treebank grammar of English. The main phrase categories in the Penn
5044 Treebank (PTB) are based on the main part-of-speech classes: noun phrase (NP), verb
5045 phrase (VP), prepositional phrase (PP), adjectival phrase (ADJP), and adverbial phrase
5046 (ADVP). The top-level category is S, which conveniently stands in for both “sentence”
5047 and the “start” symbol. **Complement clauses** (e.g., *I take the good old fashioned ground that*
5048 *the whale is a fish*) are represented by the non-terminal SBAR. The terminal symbols in
5049 the grammar are individual words, which are generated from unary productions from
5050 part-of-speech tags (the PTB tagset is described in § 8.1).

5051 This section describes some of the most common productions from the major phrase-
5052 level categories, explaining how to generate individual tag sequences. The production
5053 rules are approached in a “theory-driven” manner: first the syntactic properties of each
5054 phrase type are described, and then some of the necessary production rules are listed. But
5055 it is important to keep in mind that the Penn Treebank was produced in a “data-driven”
5056 manner. After the set of non-terminals was specified, annotators were free to analyze each
5057 sentence in whatever way seemed most linguistically accurate, subject to some high-level
5058 guidelines. The grammar of the Penn Treebank is simply the set of productions that were
5059 required to analyze the several million words of the corpus. By design, the grammar
5060 overgenerates — it does not exclude ungrammatical sentences. Furthermore, while the
5061 productions shown here cover some of the most common cases, they are only a small
5062 fraction of the several thousand different types of productions in the Penn Treebank.

5063 **Sentences**

The most common production rule for sentences is,

$$S \rightarrow NP\ VP \quad [9.28]$$

which accounts for simple sentences like *Abigail ate the kimchi* — as we will see, the direct object *the kimchi* is part of the verb phrase. But there are more complex forms of sentences as well:

$$S \rightarrow ADVP\ NP\ VP \quad \text{Unfortunately } Abigail \text{ ate the kimchi.} \quad [9.29]$$

$$S \rightarrow S\ CC\ S \quad \text{Abigail ate the kimchi and Max had a burger.} \quad [9.30]$$

$$S \rightarrow VP \quad \text{Eat the kimchi.} \quad [9.31]$$

- 5064 where ADVP is an adverbial phrase (e.g., *unfortunately*, *very unfortunately*) and CC is a
 5065 coordinating conjunction (e.g., *and*, *but*).⁸

5066 **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:

$$NP \rightarrow NN | NNS | NNP | PRP \quad [9.32]$$

$$NP \rightarrow DET\ NN | DET\ NNS | DET\ NNP \quad [9.33]$$

- 5067 The tags NN, NNS, and NNP refer to singular, plural, and proper nouns; PRP refers to
 5068 personal pronouns, and DET refers to determiners. The grammar also contains terminal
 5069 productions from each of these tags, e.g., $PRP \rightarrow I | you | we | \dots$.

Noun phrases may be modified by adjectival phrases (ADJP; e.g., *the small Russian dog*) and numbers (CD; e.g., *the five pastries*), each of which may optionally follow a determiner:

$$NP \rightarrow ADJP\ NN | ADJP\ NNS | DET\ ADJP\ NN | DET\ ADJP\ NNS \quad [9.34]$$

$$NP \rightarrow CD\ NNS | DET\ CD\ NNS | \dots \quad [9.35]$$

Some noun phrases include multiple nouns, such as *the liberation movement* and *an antelope horn*, necessitating additional productions:

$$NP \rightarrow NN\ NN | NN\ NNS | DET\ NN\ NN | \dots \quad [9.36]$$

⁸Notice that the grammar does not include the recursive production $S \rightarrow ADVP\ S$. It may be helpful to think about why this production would cause the grammar to overgenerate.

5070 These multiple noun constructions can be combined with adjectival phrases and cardinal
 5071 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] |

5072 These recursive productions are a major source of ambiguity, because the VP and PP non-
 5073 terminals can also generate NP children. Thus, the *the President of the Georgia Institute of*
 5074 *Technology* can be derived in two ways, as can *a whale taken near Shetland in October*.

5075 But aside from these few recursive productions, the noun phrase fragment of the Penn
 5076 Treebank grammar is relatively flat, containing a large of number of productions that go
 5077 from NP directly to a sequence of parts-of-speech. If noun phrases had more internal
 5078 structure, the grammar would need fewer rules, which, as we will see, would make pars-
 5079 ing faster and machine learning easier. Vadas and Curran (2011) propose to add additional
 5080 structure in the form of a new non-terminal called a **nominal modifier** (NML), e.g.,

- 5081 (9.6) a. (NP (NN crude) (NN oil) (NNS prices)) (PTB analysis)
 5082 b. (NP (NML (NN crude) (NN oil)) (NNS prices)) (NML-style analysis).

5083 Another proposal is to treat the determiner as the head of a **determiner phrase** (DP;
 5084 Abney, 1987). There are linguistic arguments for and against determiner phrases (e.g.,
 5085 Van Eynde, 2006). From the perspective of context-free grammar, DPs enable more struc-
 5086 tured analyses of some constituents, e.g.,

- 5087 (9.7) a. (NP (DT the) (JJ white) (NN whale)) (PTB analysis)
 5088 b. (DP (DT the) (NP (JJ white) (NN whale))) (DP-style analysis).

5089 Verb phrases

Verb phrases describe actions, events, and states of being. The PTB tagset distinguishes several classes of verb inflections: base form (VB; *she likes to snack*), present-tense third-person singular (VBD; *she snacks*), present tense but not third-person singular (VBP; *they snack*), past tense (VBD; *they snacked*), present participle (VBG; *they are snacking*), and past participle (VBN; *they had snacked*).⁹ Each of these forms can constitute a verb phrase on its

⁹This tagset is specific to English: for example, VBP is a meaningful category only because English morphology distinguishes third-person singular from all person-number combinations.

own:

$$\text{VP} \rightarrow \text{VB} \mid \text{VBZ} \mid \text{VBD} \mid \text{VBN} \mid \text{VBG} \mid \text{VBP} \quad [9.41]$$

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):

| | | |
|---|--------------------------------------|--------|
| $\text{VP} \rightarrow \text{MD VP}$ | <i>She will snack</i> | [9.42] |
| $\text{VP} \rightarrow \text{VBD VP}$ | <i>She had snacked</i> | [9.43] |
| $\text{VP} \rightarrow \text{VBZ VP}$ | <i>She has been snacking</i> | [9.44] |
| $\text{VP} \rightarrow \text{VBN VP}$ | <i>She has been snacking</i> | [9.45] |
| $\text{VP} \rightarrow \text{TO VP}$ | <i>She wants to snack</i> | [9.46] |
| $\text{VP} \rightarrow \text{VP CC VP}$ | <i>She buys and eats many snacks</i> | [9.47] |

- 5090 Each of these productions uses recursion, with the VP non-terminal appearing in both the
 5091 LHS and RHS. This enables the creation of complex verb phrases, such as *She will have*
 5092 *wanted to have been snacking*.

Transitive verbs take noun phrases as direct objects, and ditransitive verbs take two direct objects:

| | | |
|--|---------------------------------------|--------|
| $\text{VP} \rightarrow \text{VBZ NP}$ | <i>She teaches algebra</i> | [9.48] |
| $\text{VP} \rightarrow \text{VBG NP}$ | <i>She has been teaching algebra</i> | [9.49] |
| $\text{VP} \rightarrow \text{VBD NP NP}$ | <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:

| | | |
|---|---|--------|
| $\text{VP} \rightarrow \text{VBZ S}$ | <i>Hunter wants to eat the kimchi</i> | [9.51] |
| $\text{VP} \rightarrow \text{VBZ SBAR}$ | <i>Hunter knows that Tristan ate the kimchi</i> | [9.52] |

- 5093 The first production overgenerates, licensing sentences like **Hunter sees Tristan eats the*
 5094 *kimchi*. This problem could be addressed by designing a more specific set of sentence
 5095 non-terminals, indicating whether the main verb can be conjugated.

Verbs can also be modified by prepositional phrases and adverbial phrases:

| | | |
|---|--------------------------------|--------|
| $\text{VP} \rightarrow \text{VBZ PP}$ | <i>She studies at night</i> | [9.53] |
| $\text{VP} \rightarrow \text{VBZ ADVP}$ | <i>She studies intensively</i> | [9.54] |
| $\text{VP} \rightarrow \text{ADVP VBG}$ | <i>She is not studying</i> | [9.55] |

5096 Again, because these productions are not recursive, the grammar must include produc-
 5097 tions for every verb part-of-speech.

A special set of verbs, known as **copula**, can take **predicative adjectives** as direct ob-
 jects:

$VP \rightarrow VBZ\ ADJP$ *She is hungry* [9.56]

$VP \rightarrow VBP\ ADJP$ *Success seems increasingly unlikely* [9.57]

5098 The PTB does not have a special non-terminal for copular verbs, so this production gen-
 5099 erates non-grammatical examples such as **She eats tall*.

Particles (PRT as a phrase; RP as a part-of-speech) work to create phrasal verbs:

$VP \rightarrow VB\ PRT$ *She told them to fuck off* [9.58]

$VP \rightarrow VBD\ PRT\ NP$ *They gave up their ill-gotten gains* [9.59]

5100 As the second production shows, particle productions are required for all configurations
 5101 of verb parts-of-speech and direct objects.

5102 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, pos-
 sibly 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]

5103 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] |

5104 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> | |

5105 9.2.4 Grammatical ambiguity

5106 Context-free parsing is useful not only because it determines whether a sentence is grammatical, but mainly because the constituents and their relations can be applied to tasks such as information extraction (chapter 17) and sentence compression (Jing, 2000; Clarke and Lapata, 2008). However, the **ambiguity** of wide-coverage natural language grammars poses a serious problem for such potential applications. As an example, Figure 9.13 shows 5111 two possible analyses for the simple sentence *We eat sushi with chopsticks*, depending on 5112 whether the *chopsticks* modify *eat* or *sushi*. Realistic grammars can license thousands or 5113 even millions of parses for individual sentences. **Weighted context-free grammars** solve 5114 this problem by attaching weights to each production, and selecting the derivation with 5115 the highest score. This is the focus of chapter 10.

5116 9.3 *Mildly context-sensitive languages

5117 Beyond context-free languages lie **context-sensitive languages**, in which the expansion 5118 of a non-terminal depends on its neighbors. In the general class of context-sensitive 5119 languages, computation becomes much more challenging: the membership problem for 5120 context-sensitive languages is PSPACE-complete. Since PSPACE contains the complexity 5121 class NP (problems that can be solved in polynomial time on a non-deterministic Turing

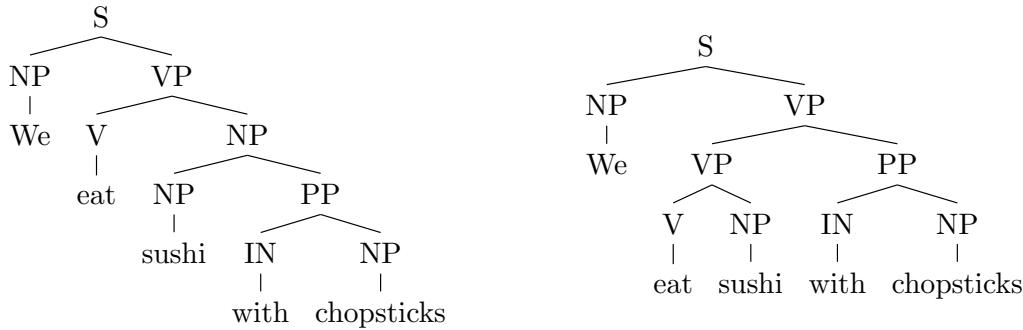


Figure 9.13: Two derivations of the same sentence

5122 machine), PSPACE-complete problems cannot be solved efficiently if $P \neq NP$. Thus, de-
 5123 signing an efficient parsing algorithm for the full class of context-sensitive languages is
 5124 probably hopeless.¹⁰

5125 However, Joshi (1985) identifies a set of properties that define **mildly context-sensitive**
 5126 **languages**, which are a strict subset of context-sensitive languages. Like context-free lan-
 5127 guages, mildly context-sensitive languages are parseable in polynomial time. However,
 5128 the mildly context-sensitive languages include non-context-free languages, such as the
 5129 “copy language” $\{ww \mid w \in \Sigma^*\}$ and the language $a^m b^n c^m d^n$. Both are characterized by
 5130 **cross-serial dependencies**, linking symbols at long distance across the string.¹¹ For exam-
 5131 ple, in the language $a^n b^m c^n d^m$, each a symbol is linked to exactly one c symbol, regardless
 5132 of the number of intervening b symbols.

5133 9.3.1 Context-sensitive phenomena in natural language

5134 Such phenomena are occasionally relevant to natural language. A classic example is found
 5135 in Swiss-German (Shieber, 1985), in which sentences such as *we let the children help Hans*
 5136 *paint the house* are realized by listing all nouns before all verbs, i.e., *we the children Hans the*
 5137 *house let help paint*. Furthermore, each noun’s determiner is dictated by the noun’s **case**
 5138 **marking** (the role it plays with respect to the verb). Using an argument that is analogous
 5139 to the earlier discussion of center-embedding (§ 9.2), Shieber describes these case marking
 5140 constraints as a set of cross-serial dependencies, homomorphic to $a^m b^n c^m d^n$, and therefore
 5141 not context-free.

¹⁰If $P \neq NP$, then it contains problems that cannot be solved in polynomial time on a non-deterministic Turing machine; equivalently, solutions to these problems cannot even be checked in polynomial time (Arora and Barak, 2009).

¹¹A further condition of the set of mildly-context-sensitive languages is *constant growth*: if the strings in the language are arranged by length, the gap in length between any pair of adjacent strings is bounded by some language specific constant. This condition excludes languages such as $\{a^{2^n} \mid n \geq 0\}$.

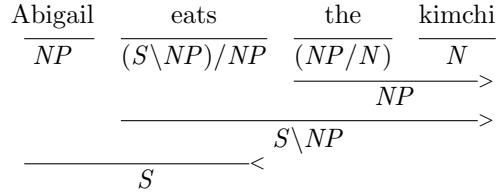


Figure 9.14: A syntactic analysis in CCG involving forward and backward function application

As with the move from regular to context-free languages, mildly context-sensitive languages can also be motivated by expedience. While finite sequences of cross-serial dependencies can in principle be handled in a context-free grammar, it is often more convenient to use a mildly context-sensitive formalism like **tree-adjoining grammar** (TAG) and **combinatory categorial grammar** (CCG). TAG-inspired parsers have been shown to be particularly effective in parsing the Penn Treebank (Collins, 1997; Carreras et al., 2008), and CCG plays a leading role in current research on semantic parsing (Zettlemoyer and Collins, 2005). These two formalisms are weakly equivalent: any language that can be specified in TAG can also be specified in CCG, and vice versa (Joshi et al., 1991). The remainder of the chapter gives a brief overview of CCG, but you are encouraged to consult Joshi and Schabes (1997) and Steedman and Baldridge (2011) for more detail on TAG and CCG respectively.

9.3.2 Combinatory categorial grammar

In combinatory categorial grammar, structural analyses are built up through a small set of generic combinatorial operations, which apply to immediately adjacent sub-structures. These operations act on the categories of the sub-structures, producing a new structure with a new category. The basic categories include S (sentence), NP (noun phrase), VP (verb phrase) and N (noun). The goal is to label the entire span of text as a sentence, S .

Complex categories, or types, are constructed from the basic categories, parentheses, and forward and backward slashes: for example, S/NP is a complex type, indicating a sentence that is lacking a noun phrase to its right; $S\backslash NP$ is a sentence lacking a noun phrase to its left. Complex types act as functions, and the most basic combinatory operations are function application to either the right or left neighbor. For example, the type of a verb phrase, such as *eats*, would be $S\backslash NP$. Applying this function to a subject noun phrase to its left results in an analysis of *Abigail eats* as category S , indicating a successful parse.

Transitive verbs must first be applied to the direct object, which in English appears to the right of the verb, before the subject, which appears on the left. They therefore have the more complex type $(S\backslash NP)/NP$. Similarly, the application of a determiner to the noun at

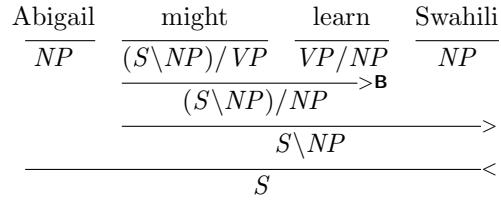


Figure 9.15: A syntactic analysis in CCG involving function composition (example modified from Steedman and Baldridge, 2011)

5171 its right results in a noun phrase, so determiners have the type NP/N. Figure 9.14 pro-
 5172 vides an example involving a transitive verb and a determiner. A key point from this
 5173 example is that it can be trivially transformed into phrase-structure tree, by treating each
 5174 function application as a constituent phrase. Indeed, when CCG's only combinatory op-
 5175 erators are forward and backward function application, it is equivalent to context-free
 5176 grammar. However, the location of the "effort" has changed. Rather than designing good
 5177 productions, the grammar designer must focus on the **lexicon** — choosing the right cate-
 5178 gories for each word. This makes it possible to parse a wide range of sentences using only
 5179 a few generic combinatory operators.

5180 Things become more interesting with the introduction of two additional operators:
 5181 **composition** and **type-raising**. Function composition enables the combination of com-
 5182 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-
 5183 ward composition).¹² Composition makes it possible to "look inside" complex types, and
 5184 combine two adjacent units if the "input" for one is the "output" for the other. Figure 9.15
 5185 shows how function composition can be used to handle modal verbs. While this sen-
 5186 tence can be parsed using only function application, the composition-based analysis is
 5187 preferable because the unit *might learn* functions just like a transitive verb, as in the exam-
 5188 ple *Abigail studies Swahili*. This in turn makes it possible to analyze conjunctions such as
 5189 *Abigail studies and might learn Swahili*, attaching the direct object *Swahili* to the entire con-
 5190 joined verb phrase *studies and might learn*. The Penn Treebank grammar fragment from
 5191 § 9.2.3 would be unable to handle this case correctly: the direct object *Swahili* could attach
 5192 only to the second verb *learn*.

5193 Type raising converts an element of type X to a more complex type: $X \Rightarrow_T T/(T\backslash X)$
 5194 (forward type-raising to type T), and $X \Rightarrow_T T\backslash(T/X)$ (backward type-raising to type
 5195 T). Type-raising makes it possible to reverse the relationship between a function and its
 5196 argument — by transforming the argument into a function over functions over arguments!
 5197 An example may help. Figure 9.15 shows how to analyze an object relative clause, *a story*
 5198 *that Abigail tells*. The problem is that *tells* is a transitive verb, expecting a direct object to
 5199 its right. As a result, *Abigail tells* is not a valid constituent. The issue is resolved by raising

¹²The subscript **B** follows notation from Curry and Feys (1958).

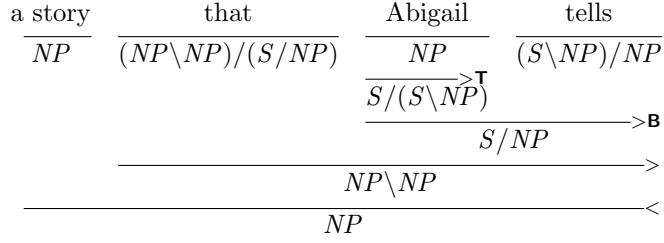


Figure 9.16: A syntactic analysis in CCG involving an object relative clause

5200 *Abigail* from NP to the complex type $(S/NP) \setminus NP$. This function can then be combined
 5201 with the transitive verb *tells* by forward composition, resulting in the type (S/NP) , which
 5202 is a sentence lacking a direct object to its right.¹³ From here, we need only design the
 5203 lexical entry for the complementizer *that* to expect a right neighbor of type (S/NP) , and
 5204 the remainder of the derivation can proceed by function application.

5205 Composition and type-raising give CCG considerable power and flexibility, but at a
 5206 price. The simple sentence *Abigail tells Max* can be parsed in two different ways: by func-
 5207 tion application (first forming the verb phrase *tells Max*), and by type-raising and compo-
 5208 sition (first forming the non-constituent *Abigail tells*). This **derivational ambiguity** does
 5209 not affect the resulting linguistic analysis, so it is sometimes known as **spurious ambi-**
 5210 **guity**. Hockenmaier and Steedman (2007) present a translation algorithm for converting
 5211 the Penn Treebank into CCG derivations, using composition and type-raising only when
 5212 necessary.

5213 Exercises

- 5214 1. Sketch out the state diagram for finite-state acceptors for the following languages
 5215 on the alphabet $\{a, b\}$.
 - 5216 a) Even-length strings. (Be sure to include 0 as an even number.)
 - 5217 b) Strings that contain *aaa* as a substring.
 - 5218 c) Strings containing an even number of *a* and an odd number of *b* symbols.
 - 5219 d) Strings in which the substring *bbb* must be terminal if it appears — the string
 5220 need not contain *bbb*, but if it does, nothing can come after it.
- 5221 2. Levenshtein edit distance is the number of insertions, substitutions, or deletions
 5222 required to convert one string to another.

¹³The missing direct object would be analyzed as a **trace** in CFG-like approaches to syntax, including the Penn Treebank.

- 5223 a) Define a finite-state acceptor that accepts all strings with edit distance 1 from
 5224 the target string, *target*.
 5225 b) Now think about how to generalize your design to accept all strings with edit
 5226 distance from the target string equal to d . If the target string has length ℓ , what
 5227 is the minimal number of states required?
- 5228 3. Construct an FSA in the style of Figure 9.3, which handles the following examples:
- 5229 • *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N
 - 5230 • *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N
- 5231 Be sure that your FSA does not accept any further derivations, such as **nationalizeral*
 5232 and **Americanizern*.
- 5233 4. Show how to construct a trigram language model in a weighted finite-state acceptor.
 5234 Make sure that you handle the edge cases at the beginning and end of the input.
- 5235 5. Extend the FST in Figure 9.6 to handle the other two parts of rule 1a of the Porter
 5236 stemmer: $-sses \rightarrow ss$, and $-ies \rightarrow -i$.
- 5237 6. § 9.1.4 describes T_O , a transducer that captures English orthography by transduc-
 5238 ing *cook + ed* \rightarrow *cooked* and *bake + ed* \rightarrow *baked*. Design an unweighted finite-state
 5239 transducer that captures this property of English orthography.
 5240 Next, augment the transducer to appropriately model the suffix *-s* when applied to
 5241 words ending in *s*, e.g. *kiss+s* \rightarrow *kisses*.
- 5242 7. Add parenthesization to the grammar in Figure 9.11 so that it is no longer ambigu-
 5243 ous.
- 5244 8. Construct three examples — a noun phrase, a verb phrase, and a sentence — which
 5245 can be derived from the Penn Treebank grammar fragment in § 9.2.3, yet are not
 5246 grammatical. Avoid reusing examples from the text. Optionally, propose corrections
 5247 to the grammar to avoid generating these cases.
- 5248 9. Produce parses for the following sentences, using the Penn Treebank grammar frag-
 5249 ment from § 9.2.3.
- 5250 (9.8) This aggression will not stand.
 5251 (9.9) I can get you a toe.
 5252 (9.10) Sometimes you eat the bar and sometimes the bar eats you.

5253 Then produce parses for three short sentences from a news article from this week.

5254 10. * One advantage of CCG is its flexibility in handling coordination:

5255 (9.11) a. *Hunter and Tristan speak Hawaiian*

5256 b. *Hunter speaks and Tristan understands Hawaiian*

Define the lexical entry for *and* as

$$\textit{and} := (X/X) \setminus X, \quad [9.77]$$

5257 where X can refer to any type. Using this lexical entry, show how to parse the two
5258 examples above. In the second example, *Swahili* should be combined with the coor-
5259 dination *Abigail speaks and Max understands*, and not just with the verb *understands*.

5260 **Chapter 10**

5261 **Context-free parsing**

5262 Parsing is the task of determining whether a string can be derived from a given context-
5263 free grammar, and if so, how. A parser’s output is a tree, like the ones shown in Fig-
5264 ure 9.13. Such trees can answer basic questions of who-did-what-to-whom, and have ap-
5265 plications in downstream tasks like semantic analysis (chapter 12 and 13) and information
5266 extraction (chapter 17).

For a given input and grammar, how many parse trees are there? Consider a minimal context-free grammar with only one non-terminal, X , and the following productions:

$$\begin{aligned} X &\rightarrow X \ X \\ X &\rightarrow aardvark \mid abacus \mid \dots \mid zyther \end{aligned}$$

The second line indicates unary productions to every nonterminal in Σ . In this grammar, the number of possible derivations for a string w is equal to the number of binary bracketings, e.g.,

$$(((w_1 w_2) w_3) w_4) w_5), \quad (((w_1 (w_2 w_3)) w_4) w_5), \quad ((w_1 (w_2 (w_3 w_4))) w_5), \quad \dots$$

5267 The number of such bracketings is a **Catalan number**, which grows super-exponentially
5268 in the length of the sentence, $C_n = \frac{(2n)!}{(n+1)n!}$. As with sequence labeling, it is only possible to
5269 exhaustively search the space of parses by resorting to locality assumptions, which make it
5270 possible to search efficiently by reusing shared substructures with dynamic programming.
5271 This chapter focuses on a bottom-up dynamic programming algorithm, which enables
5272 exhaustive search of the space of possible parses, but imposes strict limitations on the
5273 form of scoring function. These limitations can be relaxed by abandoning exhaustive
5274 search. Non-exact search methods will be briefly discussed at the end of this chapter, and
5275 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

5276 **10.1 Deterministic bottom-up parsing**

5277 The **CKY algorithm**¹ is a bottom-up approach to parsing in a context-free grammar. It
 5278 efficiently tests whether a string is in a language, without enumerating all possible parses.
 5279 The algorithm first forms small constituents, and then tries to merge them into larger
 5280 constituents.

5281 To understand the algorithm, consider the input, *We eat sushi with chopsticks*. According-
 5282 ing to the toy grammar in Table 10.1, each terminal symbol can be generated by exactly
 5283 one unary production, resulting in the sequence NP V NP IN NP. In real examples, there
 5284 may be many unary productions for each individual token. In any case, the next step
 5285 is to try to apply binary productions to merge adjacent symbols into larger constituents:
 5286 for example, V NP can be merged into a verb phrase (VP), and IN NP can be merged
 5287 into a prepositional phrase (PP). Bottom-up parsing searches for a series of mergers that
 5288 ultimately results in the start symbol S covering the entire input.

5289 The CKY algorithm systematizes this search by incrementally constructing a table t in
 5290 which each cell $t[i, j]$ contains the set of nonterminals that can derive the span $w_{i+1:j}$. The
 5291 algorithm fills in the upper right triangle of the table; it begins with the diagonal, which
 5292 corresponds to substrings of length 1, and then computes derivations for progressively
 5293 larger substrings, until reaching the upper right corner $t[0, M]$, which corresponds to the
 5294 entire input, $w_{1:M}$. If the start symbol S is in $t[0, M]$, then the string w is in the language
 5295 defined by the grammar. This process is detailed in Algorithm 13, and the resulting data
 5296 structure is shown in Figure 10.1. Informally, here's how it works:

- 5297 • Begin by filling in the diagonal: the cells $t[m - 1, m]$ for all $m \in \{1, 2, \dots, M\}$. These
 5298 cells are filled with terminal productions that yield the individual tokens; for the
 5299 word $w_2 = \text{sushi}$, we fill in $t[1, 2] = \{\text{NP}\}$, and so on.
- 5300 • Then fill in the next diagonal, in which each cell corresponds to a subsequence of
 5301 length two: $t[0, 2], t[1, 3], \dots, t[M - 2, M]$. These cells are filled in by looking for

¹The name is for Cocke-Kasami-Younger, the inventors of the algorithm. It is a special case of **chart parsing**, because its stores reusable computations in a chart-like data structure.

binary productions capable of producing at least one entry in each of the cells corresponding to left and right children. For example, VP can be placed in the cell $t[1, 3]$ because the grammar includes the production $VP \rightarrow V\ NP$, and because the chart contains $V \in t[1, 2]$ and $NP \in t[2, 3]$.

- At the next diagonal, the entries correspond to spans of length three. At this level, there is an additional decision at each cell: where to split the left and right children. The cell $t[i, j]$ corresponds to the subsequence $w_{i+1:j}$, and we must choose some *split point* $i < k < j$, so that the span $w_{i+1:k}$ is the left child, and the span $w_{k+1:j}$ is the right child. We consider all possible k , looking for productions that generate elements in $t[i, k]$ and $t[k, j]$; the left-hand side of all such productions can be added to $t[i, j]$. When it is time to compute $t[i, j]$, the cells $t[i, k]$ and $t[k, j]$ are guaranteed to be complete, since these cells correspond to shorter sub-strings of the input.
- The process continues until we reach $t[0, M]$.

Figure 10.1 shows the chart that arises from parsing the sentence *We eat sushi with chopsticks* using the grammar defined above.

10.1.1 Recovering the parse tree

As with the Viterbi algorithm, it is possible to identify a successful parse by storing and traversing an additional table of back-pointers. If we add an entry X to cell $t[i, j]$ by using the production $X \rightarrow YZ$ and the split point k , then we store the back-pointer $b[i, j, X] = (Y, Z, k)$. Once the table is complete, we can recover a parse by tracing these pointers, starting at $b[0, M, S]$, and stopping when they ground out at terminal productions.

For ambiguous sentences, there will be multiple paths to reach $S \in t[0, M]$. For example, in Figure 10.1, the goal state $S \in t[0, M]$ is reached through the state $VP \in t[1, 5]$, and there are two different ways to generate this constituent: one with *(eat sushi)* and *(with chopsticks)* as children, and another with *(eat)* and *(sushi with chopsticks)* as children. The presence of multiple paths indicates that the input can be generated by the grammar in more than one way. In Algorithm 13, one of these derivations is selected arbitrarily. As discussed in § 10.3, **weighted context-free grammars** compute a score for all permissible derivations, and a minor modification of CKY allows it to identify the single derivation with the maximum score.

10.1.2 Non-binary productions

As presented above, the CKY algorithm assumes that all productions with non-terminals on the right-hand side (RHS) are binary. In real grammars, such as the one considered in chapter 9, there are other types of productions: some have more than two elements on the right-hand side, and others produce a single non-terminal.

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). 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  $X \rightarrow (\text{TRACEBACK}(Y, i, k, b), \text{TRACEBACK}(Z, k, j, b))$ 

```

- 5337 • Productions with more than two elements on the right-hand side can be **binarized**
5338 by creating additional non-terminals, as described in § 9.2.1. For example, the pro-
5339 duction $VP \rightarrow V NP NP$ (for ditransitive verbs) can be converted to $VP \rightarrow VP_{ditrans}/NP NP$,
5340 by adding the non-terminal $VP_{ditrans}/NP$ and the production $VP_{ditrans}/NP \rightarrow V NP$.
- 5341 • What about unary productions like $VP \rightarrow V$? While such productions are not a
5342 part of Chomsky Normal Form — and can therefore be eliminated in preprocessing
5343 the grammar — in practice, a more typical solution is to modify the CKY algorithm.
5344 The algorithm makes a second pass on each diagonal in the table, augmenting each
5345 cell $t[i, j]$ with all possible unary productions capable of generating each item al-
5346 ready in the cell: formally, $t[i, j]$ is extended to its **unary closure**. Suppose the ex-
5347 ample grammar in Table 10.1 were extended to include the production $VP \rightarrow V$,
5348 enabling sentences with intransitive verb phrases, like *we eat*. Then the cell $t[1, 2]$
5349 — corresponding to the word *eat* — would first include the set $\{V\}$, and would be
5350 augmented to the set $\{V, VP\}$ during this second pass.

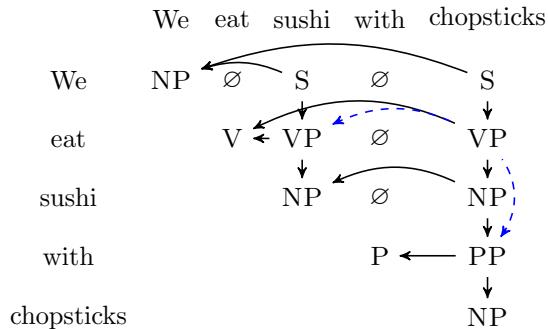


Figure 10.1: An example completed CKY chart. The solid and dashed lines show the back pointers resulting from the two different derivations of VP in position $t[1, 5]$.

5351 10.1.3 Complexity

5352 For an input of length M and a grammar with R productions and N non-terminals, the
 5353 space complexity of the CKY algorithm is $\mathcal{O}(M^2N)$: the number of cells in the chart is
 5354 $\mathcal{O}(M^2)$, and each cell must hold $\mathcal{O}(N)$ elements. The time complexity is $\mathcal{O}(M^3R)$: each
 5355 cell is computed by searching over $\mathcal{O}(M)$ split points, with R possible productions for
 5356 each split point. Both the time and space complexity are considerably worse than the
 5357 Viterbi algorithm, which is linear in the length of the input.

5358 10.2 Ambiguity

5359 In natural language, there is rarely a single parse for a given sentence. The main culprit is
 5360 ambiguity, which is endemic to natural language syntax. Here are a few broad categories:

- 5361 • **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
 5362 or the direct object.
- 5364 • **Modifier scope:** e.g., *southern food store, plastic cup holder*. In these examples, the first
 5365 word could be modifying the subsequent adjective, or the final noun.
- 5366 • **Particle versus preposition:** e.g., *The puppy tore up the staircase*. Phrasal verbs like
 5367 *tore up* often include particles which could also act as prepositions. This has struc-
 5368 tural implications: if *up* is a preposition, then *up the staircase* is a prepositional
 5369 phrase; if *up* is a particle, then *the staircase* is the direct object to the verb.
- 5370 • **Complement structure:** e.g., *The students complained to the professor that they didn't
 5371 understand*. This is another form of attachment ambiguity, where the complement

5372 *that they didn't understand* could attach to the main verb (*complained*), or to the indi-
 5373 rect object (*the professor*).

- 5374 • **Coordination scope:** e.g., “I see,” said the blind man, as he picked up the hammer and
 5375 saw. In this example, the lexical ambiguity for *saw* enables it to be coordinated either
 5376 with the noun *hammer* or the verb *picked up*.

5377 These forms of ambiguity can combine, so that seemingly simple headlines like *Fed*
 5378 *raises interest rates* have dozens of possible analyses even in a minimal grammar. In a
 5379 broad coverage grammar, typical sentences can have millions of parses. While careful
 5380 grammar design can chip away at this ambiguity, a better strategy is combine broad cov-
 5381 erage parsers with data-driven strategies for identifying the correct analysis.

5382 10.2.1 Parser evaluation

5383 Before continuing to parsing algorithms that are able to handle ambiguity, let us stop
 5384 to consider how to measure parsing performance. Suppose we have a set of *reference*
 5385 *parses* — the ground truth — and a set of *system parses* that we would like to score. A
 5386 simple solution would be per-sentence accuracy: the parser is scored by the proportion of
 5387 sentences on which the system and reference parses exactly match.² But as any student
 5388 knows, it always nice to get *partial credit*, which we can assign to analyses that correctly
 5389 match parts of the reference parse. The PARSEval metrics (Grishman et al., 1992) score
 5390 each system parse via:

5391 **Precision:** the fraction of constituents in the system parse that match a constituent in the
 5392 reference parse.

5393 **Recall:** the fraction of constituents in the reference parse that match a constituent in the
 5394 system parse.

5395 In **labeled precision** and **recall**, the system must also match the phrase type for each
 5396 constituent; in **unlabeled precision** and **recall**, it is only required to match the constituent
 5397 structure. As described in chapter 4, the precision and recall can be combined into an
 5398 *F*-MEASURE by their harmonic mean.

5399 Suppose that the left tree of Figure 10.2 is the system parse, and that the right tree is
 5400 the reference parse. Then:

- 5401 • $S \rightarrow w_{1:5}$ is *true positive*, because it appears in both trees.

²Most parsing papers do not report results on this metric, but Suzuki et al. (2018) find that a strong parser recovers the exact parse in roughly 50% of all sentences. Performance on short sentences is generally much higher.

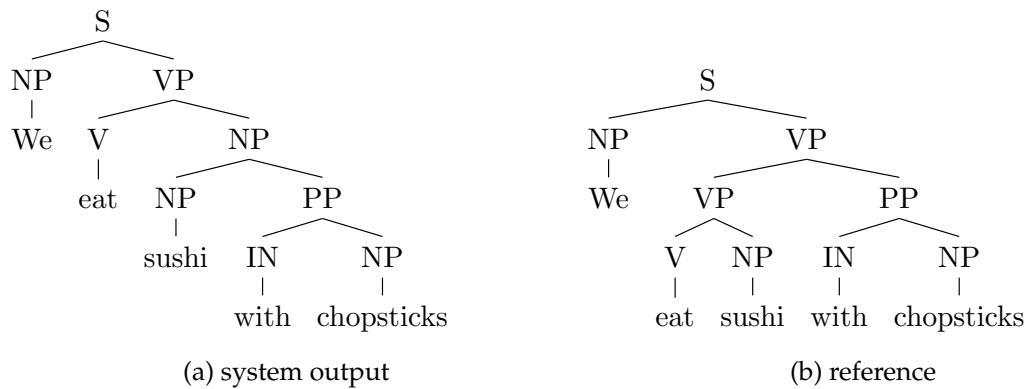


Figure 10.2: Two possible analyses from the grammar in Table 10.1

- VP → $w_{2:5}$ is *true positive* as well.
 - NP → $w_{3:5}$ is *false positive*, because it appears only in the system output.
 - PP → $w_{4:5}$ is *true positive*, because it appears in both trees.
 - VP → $w_{2:3}$ is *false negative*, because it appears only in the reference.

The labeled and unlabeled precision of this parse is $\frac{3}{4} = 0.75$, and the recall is $\frac{3}{4} = 0.75$, for an F-measure of 0.75. For an example in which precision and recall are not equal, suppose the reference parse instead included the production $VP \rightarrow V NP PP$. In this parse, the reference does not contain the constituent $w_{2,3}$, so the recall would be 1.³

5410 10.2.2 Local solutions

5411 Some ambiguity can be resolved locally. Consider the following examples,

- 5412 (10.1) a. We met the President on Monday.
5413 b. We met the President of Mexico.

Each case ends with a prepositional phrase, which can be attached to the verb *met* or the noun phrase *the president*. If given a labeled corpus, we can compare the likelihood of the observing the preposition alongside each candidate attachment point,

$$p(on \mid met) \geq p(on \mid President) \quad [10.1]$$

$$p(of \mid met) \geq p(of \mid President). \quad [10.2]$$

³While the grammar must be binarized before applying the CKY algorithm, evaluation is performed on the original parses. It is therefore necessary to “unbinarize” the output of a CKY-based parser, converting it back to the original grammar.

5414 A comparison of these probabilities would successfully resolve this case (Hindle and
 5415 Rooth, 1993). Other cases, such as the example *we eat sushi with chopsticks*, require con-
 5416 sidering the object of the preposition: consider the alternative *we eat sushi with soy sauce*.
 5417 With sufficient labeled data, some instances of attachment ambiguity can be solved by
 5418 supervised classification (Ratnaparkhi et al., 1994).

5419 However, there are inherent limitations to local solutions. While toy examples may
 5420 have just a few ambiguities to resolve, realistic sentences have thousands or millions of
 5421 possible parses. Furthermore, attachment decisions are interdependent, as shown in the
 5422 garden path example:

5423 (10.2) Cats scratch people with claws with knives.

5424 We may want to attach *with claws* to *scratch*, as would be correct in the shorter sentence
 5425 in *cats scratch people with claws*. But this leaves nowhere to attach *with knives*. The cor-
 5426 rect interpretation can be identified only by considering the attachment decisions jointly.
 5427 The huge number of potential parses may seem to make exhaustive search impossible.
 5428 But as with sequence labeling, locality assumptions make it possible to search this space
 5429 efficiently.

5430 10.3 Weighted Context-Free Grammars

5431 Let us define a derivation τ as a set of **anchored productions**,

$$\tau = \{X \rightarrow \alpha, (i, j, k)\}, \quad [10.3]$$

5432 with X corresponding to the left-hand side non-terminal and α corresponding to the right-
 5433 hand side. For grammars in Chomsky normal form, α is either a pair of non-terminals or
 5434 a terminal symbol. The indices i, j, k anchor the production in the input, with X deriving
 5435 the span $w_{i+1:j}$. For binary productions, $w_{i+1:k}$ indicates the span of the left child, and
 5436 $w_{k+1:j}$ indicates the span of the right child; for unary productions, k is ignored. For an
 5437 input w , the optimal parse is,

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(w)}{\operatorname{argmax}} \Psi(\tau), \quad [10.4]$$

5438 where $\mathcal{T}(w)$ is the set of derivations that yield the input w .

5439 Define a scoring function Ψ that decomposes across anchored productions,

$$\Psi(\tau) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \psi(X \rightarrow \alpha, (i, j, k)). \quad [10.5]$$

5440 This is a locality assumption, akin to the assumption in Viterbi sequence labeling. In this
 5441 case, the assumption states that the overall score is a sum over scores of productions,

| | | $\psi(\cdot)$ | $\exp \psi(\cdot)$ |
|----|---------------------------------|---------------|--------------------|
| 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.

5442 which are computed independently. In a **weighted context-free grammar** (WCFG), the
 5443 score of each anchored production $X \rightarrow (\alpha, (i, j, k))$ is simply $\psi(X \rightarrow \alpha)$, ignoring the
 5444 anchor (i, j, k) . In other parsing models, the anchors can be used to access features of the
 5445 input, while still permitting efficient bottom-up parsing.

Example Consider the weighted grammar shown in Table 10.2, and the analysis in Figure 10.2b.

$$\begin{aligned} \Psi(\tau) = & \psi(S \rightarrow \text{NP VP}) + \psi(VP \rightarrow \text{VP PP}) + \psi(VP \rightarrow \text{V NP}) + \psi(PP \rightarrow \text{IN NP}) \\ & + \psi(NP \rightarrow \text{We}) + \psi(V \rightarrow \text{eat}) + \psi(NP \rightarrow \text{sushi}) + \psi(IN \rightarrow \text{with}) + \psi(NP \rightarrow \text{chopsticks}) \end{aligned} \quad [10.6]$$

$$= 0 - 2 - 1 + 0 - 2 + 0 - 3 + 0 - 3 = -11. \quad [10.7]$$

5446 In the alternative parse in Figure 10.2a, the production $VP \rightarrow \text{VP PP}$ (with score -2) is
 5447 replaced with the production $NP \rightarrow \text{NP PP}$ (with score -1); all other productions are the
 5448 same. As a result, the score for this parse is -10. This example hints at a problem with
 5449 WCFG parsing on non-terminals such as NP, VP, and PP: a WCFG will *always* prefer
 5450 either VP or NP attachment, regardless of what is being attached! Solutions to this issue
 5451 are discussed in § 10.5.

Algorithm 14 CKY algorithm for parsing a string $w \in \Sigma^*$ in a weighted context-free grammar (N, Σ, R, S) , where N is the set of non-terminals and R is the set of weighted productions. The grammar is assumed to be in Chomsky normal form (§ 9.2.1). 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( $S, 0, M, b$ )

```

5452 **10.3.1 Parsing with weighted context-free grammars**

5453 The optimization problem in Equation 10.4 can be solved by modifying the CKY algo-
 5454 rithm. In the deterministic CKY algorithm, each cell $t[i, j]$ stored a set of non-terminals
 5455 capable of deriving the span $w_{i+1:j}$. We now augment the table so that the cell $t[i, j, X]$
 5456 is the *score of the best derivation* of $w_{i+1:j}$ from non-terminal X . This score is computed
 5457 recursively: for the anchored binary production $(X \rightarrow Y Z, (i, j, k))$, we compute:

- 5458 • the score of the anchored production, $\psi(X \rightarrow Y Z, (i, j, k))$;
- 5459 • the score of the best derivation of the left child, $t[i, k, Y]$;
- 5460 • the score of the best derivation of the right child, $t[k, j, Z]$.

5461 These scores are combined by addition. As in the unweighted CKY algorithm, the table
 5462 is constructed by considering spans of increasing length, so the scores for spans $t[i, k, Y]$
 5463 and $t[k, j, Z]$ are guaranteed to be available at the time we compute the score $t[i, j, X]$. The
 5464 value $t[0, M, S]$ is the score of the best derivation of w from the grammar. Algorithm 14
 5465 formalizes this procedure.

5466 As in unweighted CKY, the parse is recovered from the table of back pointers b , where
 5467 each $b[i, j, X]$ stores the argmax split point k and production $X \rightarrow Y Z$ in the derivation of
 5468 $w_{i+1:j}$ from X . The top scoring parse can be obtained by tracing these pointers backwards
 5469 from $b[0, M, S]$, all the way to the terminal symbols. This is analogous to the computation

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

```

5470 of the best sequence of labels in the Viterbi algorithm by tracing pointers backwards from
 5471 the end of the trellis. Note that we need only store back-pointers for the *best* path to
 5472 $t[i, j, X]$; this follows from the locality assumption that the global score for a parse is a
 5473 combination of the local scores of each production in the parse.

Example Let's revisit the parsing table in Figure 10.1. In a weighted CFG, each cell would include a score for each non-terminal; non-terminals that cannot be generated are assumed to have a score of $-\infty$. The first diagonal contains the scores of unary productions: $t[0, 1, \text{NP}] = -2$, $t[1, 2, \text{V}] = 0$, and so on. The next diagonal contains 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,

$$t[1, 5, \text{VP}] = \max(\psi(\text{VP} \rightarrow \text{VP PP}, (1, 3, 5)) + t[1, 3, \text{VP}] + t[3, 5, \text{PP}], \\ \psi(\text{VP} \rightarrow \text{V NP}, (1, 2, 5)) + t[1, 2, \text{V}] + t[2, 5, \text{NP}]) \quad [10.8]$$

$$= \max(-2 - 4 - 3, -1 + 0 - 7) = -8. \quad [10.9]$$

5474 Since the second case is the argmax, we set the back-pointer $b[1, 5, \text{VP}] = (\text{V}, \text{NP}, 2)$, enabling
 5475 the optimal derivation to be recovered.

5476 10.3.2 Probabilistic context-free grammars

5477 **Probabilistic context-free grammars (PCFGs)** are a special case of weighted context-
 5478 free grammars that arises when the weights correspond to probabilities. Specifically, the
 5479 weight $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha | X)$, where the probability of the right-hand side
 5480 α is conditioned on the non-terminal X . These probabilities must be normalized over all
 5481 possible right-hand sides, so that $\sum_\alpha p(\alpha | X) = 1$, for all X . For a given parse τ , the prod-
 5482 uct of the probabilities of the productions is equal to $p(\tau)$, under the **generative model**
 5483 $\tau \sim \text{DRAWSUBTREE}(S)$, where the function DRAWSUBTREE is defined in Algorithm 15.

5484 The conditional probability of a parse given a string is,

$$p(\tau \mid \mathbf{w}) = \frac{p(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} p(\tau')} = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau')}, \quad [10.10]$$

5485 where $\Psi(\tau) = \sum_{X \rightarrow \alpha, (i,j,k) \in \tau} \psi(X \rightarrow \alpha)$; the anchor is ignored. Because the probability
 5486 is monotonic in the score $\Psi(\tau)$, the maximum likelihood parse can be identified by the
 5487 CKY algorithm without modification. If a normalized probability $p(\tau \mid \mathbf{w})$ is required,
 5488 the denominator of Equation 10.10 can be computed by the **inside recurrence**, described
 5489 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 $p(\tau_1 \mid \mathbf{w})$ is equal to,

$$p(\tau_1 \mid \mathbf{w}) = \frac{p(\tau_1)}{p(\tau_1) + p(\tau_2)} = \frac{\exp \Psi(\tau_1)}{\exp \Psi(\tau_1) + \exp \Psi(\tau_2)} = \frac{2^{-10}}{2^{-10} + 2^{-11}} = \frac{2}{3}. \quad [10.11]$$

5490 **The inside recurrence** The denominator of Equation 10.10 can be viewed as a language
 5491 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:

$$t[i, j, X] = \log \sum_{k, Y, Z} \exp (\psi(X \rightarrow Y Z) + t[i, k, Y] + t[k, j, Z]) \quad [10.13]$$

$$= \log \sum_{k, Y, Z} \exp (\log p(Y Z \mid X) + \log p(Y \rightarrow \mathbf{w}_{i+1:k}) + \log p(Z \rightarrow \mathbf{w}_{k+1:j})) \quad [10.14]$$

$$= \log \sum_{k, Y, Z} p(Y Z \mid X) \times p(Y \rightarrow \mathbf{w}_{i+1:k}) \times p(Z \rightarrow \mathbf{w}_{k+1:j}) \quad [10.15]$$

$$= \log \sum_{k, Y, Z} p(Y Z, \mathbf{w}_{i+1:k}, \mathbf{w}_{k+1:j} \mid X) \quad [10.16]$$

$$= \log p(X \rightsquigarrow \mathbf{w}_{i+1:j}), \quad [10.17]$$

5492 with $X \rightsquigarrow w_{i+1:j}$ indicating the event that non-terminal X yields the tokens $(w_{i+1}, w_{i+2}, \dots, w_j)$.
 5493 The recursive computation of $t[i, j, X]$ is called the **inside recurrence**, because it computes
 5494 the probability of each subtree as a combination of the probabilities of the smaller subtrees
 5495 that are inside of it. The name implies a corresponding **outside recurrence**, which com-
 5496 putes the probability of a non-terminal X spanning $w_{i+1:j}$, joint with the outside context
 5497 $(w_{1:i}, w_{j+1:M})$. This recurrence is described in § 10.4.3. The inside and outside recurrences
 5498 are analogous to the forward and backward recurrences in probabilistic sequence label-
 5499 ing (see § 7.5.3). They can be used to compute the marginal probabilities of individual
 5500 anchored productions, $p(X \rightarrow \alpha, (i, j, k) \mid \mathbf{w})$, summing over all possible derivations of
 5501 \mathbf{w} .

5502 **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:

$$t[i, j, X] = \bigoplus_{k, Y, Z} \psi(X \rightarrow Y Z, (i, j, k)) \otimes t[i, k, Y] \otimes t[k, j, Z]. \quad [10.18]$$

5503 This recurrence subsumes all of the algorithms that have been discussed in this chapter to
 5504 this point.

5505 **Unweighted CKY.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a *Boolean truth value* $\{\top, \perp\}$, \otimes is logical
 5506 conjunction, and \bigoplus is logical disjunction, then we derive CKY recurrence for un-
 5507 weighted context-free grammars, discussed in § 10.1 and Algorithm 13.

5508 **Weighted CKY.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a scalar score, \otimes is addition, and \bigoplus is maxi-
 5509 mization, then we derive the CKY recurrence for weighted context-free grammars,
 5510 discussed in § 10.3 and Algorithm 14. When $\psi(X \rightarrow \alpha, (i, j, k)) = \log p(\alpha \mid X)$,
 5511 this same setting derives the CKY recurrence for finding the maximum likelihood
 5512 derivation in a probabilistic context-free grammar.

5513 **Inside recurrence.** When $\psi(X \rightarrow \alpha, (i, j, k))$ is a log probability, \otimes is addition, and $\bigoplus =$
 5514 $\log \sum \exp$, then we derive the inside recurrence for probabilistic context-free gram-
 5515 mmars, discussed in § 10.3.2. It is also possible to set $\psi(X \rightarrow \alpha, (i, j, k))$ directly equal
 5516 to the probability $p(\alpha \mid X)$. In this case, \otimes is multiplication, and \bigoplus is addition.
 5517 While this may seem more intuitive than working with log probabilities, there is the
 5518 risk of underflow on long inputs.

5519 Regardless of how the scores are combined, the key point is the locality assumption:
 5520 the score for a derivation is the combination of the independent scores for each anchored

5521 production, and these scores do not depend on any other part of the derivation. For exam-
 5522 ple, if two non-terminals are siblings, the scores of productions from these non-terminals
 5523 are computed independently. This locality assumption is analogous to the first-order
 5524 Markov assumption in sequence labeling, where the score for transitions between tags
 5525 depends only on the previous tag and current tag, and not on the history. As with se-
 5526 quence labeling, this assumption makes it possible to find the optimal parse efficiently; its
 5527 linguistic limitations are discussed in § 10.5.

5528 10.4 Learning weighted context-free grammars

5529 Like sequence labeling, context-free parsing is a form of structure prediction. As a result,
 5530 WCFGs can be learned using the same set of algorithms: generative probabilistic models,
 5531 structured perceptron, maximum conditional likelihood, and maximum margin learning.
 5532 In all cases, learning requires a **treebank**, which is a dataset of sentences labeled with
 5533 context-free parses. Parsing research was catalyzed by the **Penn Treebank** (Marcus et al.,
 5534 1993), the first large-scale dataset of this type (see § 9.2.2). Phrase structure treebanks exist
 5535 for roughly two dozen other languages, with coverage mainly restricted to European and
 5536 East Asian languages, plus Arabic and Urdu.

5537 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:

$$\psi(X \rightarrow \alpha) = \log p(X \rightarrow \alpha) \quad [10.19]$$

$$\hat{p}(X \rightarrow \alpha) = \frac{\text{count}(X \rightarrow \alpha)}{\text{count}(X)}. \quad [10.20]$$

5538 For example, the probability of the production $\text{NP} \rightarrow \text{DET NN}$ is the corpus count of
 5539 this production, divided by the count of the non-terminal NP. This estimator applies
 5540 to terminal productions as well: the probability of $\text{NN} \rightarrow \text{whale}$ is the count of how often
 5541 *whale* appears in the corpus as generated from an NN tag, divided by the total count of the
 5542 NN tag. Even with the largest treebanks — currently on the order of one million tokens
 5543 — it is difficult to accurately compute probabilities of even moderately rare events, such
 5544 as $\text{NN} \rightarrow \text{whale}$. Therefore, smoothing is critical for making PCFGs effective.

5545 **10.4.2 Feature-based parsing**

5546 The scores for each production can be computed as an inner product of weights and fea-
 5547 tures,

$$\psi(X \rightarrow \alpha, (i, j, k)) = \boldsymbol{\theta} \cdot \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}), \quad [10.21]$$

5548 where the feature vector \mathbf{f} is a function of the left-hand side X , the right-hand side α , the
 5549 anchor indices (i, j, k) , and the input \mathbf{w} .

5550 The basic feature $\mathbf{f}(X, \alpha, (i, j, k)) = \{(X, \alpha)\}$ encodes only the identity of the produc-
 5551 tion itself. This gives rise to a discriminatively-trained model with the same expressive-
 5552 ness as a PCFG. Features on anchored productions can include the words that border the
 5553 span w_i, w_{j+1} , the word at the split point w_{k+1} , the presence of a verb or noun in the left
 5554 child span $w_{i+1:k}$, and so on (Durrett and Klein, 2015). Scores on anchored productions
 5555 can be incorporated into CKY parsing without any modification to the algorithm, because
 5556 it is still possible to compute each element of the table $t[i, j, X]$ recursively from its imme-
 5557 diate children.

5558 Other features can be obtained by grouping elements on either the left-hand or right-
 5559 hand side: for example it can be particularly beneficial to compute additional features
 5560 by clustering terminal symbols, with features corresponding to groups of words with
 5561 similar syntactic properties. The clustering can be obtained from unlabeled datasets that
 5562 are much larger than any treebank, improving coverage. Such methods are described in
 5563 chapter 14.

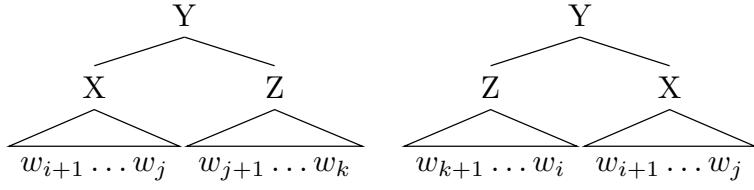
Feature-based parsing models can be estimated using the usual array of discriminative learning techniques. For example, a structure perceptron update can be computed as (Carreras et al., 2008),

$$\mathbf{f}(\tau, \mathbf{w}^{(i)}) = \sum_{(X \rightarrow \alpha, (i, j, k)) \in \tau} \mathbf{f}(X, \alpha, (i, j, k), \mathbf{w}^{(i)}) \quad [10.22]$$

$$\hat{\tau} = \underset{\tau \in \mathcal{T}(\mathbf{w})}{\operatorname{argmax}} \boldsymbol{\theta} \cdot \mathbf{f}(\tau, \mathbf{w}^{(i)}) \quad [10.23]$$

$$\boldsymbol{\theta} \leftarrow \mathbf{f}(\tau^{(i)}, \mathbf{w}^{(i)}) - \mathbf{f}(\hat{\tau}, \mathbf{w}^{(i)}). \quad [10.24]$$

5564 A margin-based objective can be optimized by selecting $\hat{\tau}$ through cost-augmented decod-
 5565 ing (§ 2.4.2), enforcing a margin of $\Delta(\hat{\tau}, \tau)$ between the hypothesis and the reference parse,
 5566 where Δ is a non-negative cost function, such as the Hamming loss (Stern et al., 2017). It
 5567 is also possible to train feature-based parsing models by conditional log-likelihood, as
 5568 described in the next section.

Figure 10.3: The two cases faced by the outside recurrence in the computation of $\beta(i, j, X)$

5569 **10.4.3 *Conditional random field parsing**

5570 The score of a derivation $\Psi(\tau)$ can be converted into a probability by normalizing over all
5571 possible derivations,

$$p(\tau \mid \mathbf{w}) = \frac{\exp \Psi(\tau)}{\sum_{\tau' \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau')}. \quad [10.25]$$

5572 Using this probability, a WCFG can be trained by maximizing the conditional log-likelihood
5573 of a labeled corpus.

5574 Just as in logistic regression and the conditional random field over sequences, the
5575 gradient of the conditional log-likelihood is the difference between the observed and ex-
5576 pected counts of each feature. The expectation $E_{\tau \mid \mathbf{w}}[\mathbf{f}(\tau, \mathbf{w}^{(i)}; \boldsymbol{\theta})]$ requires summing over
5577 all possible parses, and computing the marginal probabilities of anchored productions,
5578 $p(X \rightarrow \alpha, (i, j, k) \mid \mathbf{w})$. In CRF sequence labeling, marginal probabilities over tag bigrams
5579 are computed by the two-pass **forward-backward algorithm** (§ 7.5.3). The analogue for
5580 context-free grammars is the **inside-outside algorithm**, in which marginal probabilities
5581 are computed from terms generated by an upward and downward pass over the parsing
5582 chart:

- 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,

$$\alpha(i, j, X) \triangleq \log \sum_{(X \rightarrow Y \ Z)} \sum_{k=i+1}^j \exp (\psi(X \rightarrow Y \ Z, (i, j, k)) + \alpha(i, k, Y) + \alpha(k, j, Z)). \quad [10.26]$$

5583 The initial condition of this recurrence is $\alpha(m - 1, m, X) = \psi(X \rightarrow w_m)$. The de-
5584 nominator $\sum_{\tau \in \mathcal{T}(\mathbf{w})} \exp \Psi(\tau)$ is equal to $\exp \alpha(0, M, S)$.

- The downward pass is performed by the **outside recurrence**, which recursively pop-
ulates 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,

$$\exp \beta(i, j, X) \triangleq \sum_{(Y \rightarrow X \mid Z)} \sum_{k=j+1}^M \exp [\psi(Y \rightarrow X \mid Z, (i, k, j)) + \alpha(j, k, Z) + \beta(i, k, Y)] \quad [10.27]$$

$$+ \sum_{(Y \rightarrow Z \mid X)} \sum_{k=0}^{i-1} \exp [\psi(Y \rightarrow Z \mid X, (k, i, j)) + \alpha(k, i, Z) + \beta(k, j, Y)]. \quad [10.28]$$

5585 The first line of Equation 10.28 is the score under the condition that X is a left child
 5586 of its parent, which spans $w_{i+1:k}$, with $k > j$; the second line is the score under the
 5587 condition that X is a right child of its parent Y , which spans $w_{k+1:j}$, with $k < i$.
 5588 The two cases are shown in Figure 10.3. In each case, we sum over all possible
 5589 productions with X on the right-hand side. The parent Y is bounded on one side
 5590 by either i or j , depending on whether X is a left or right child of Y ; we must sum
 5591 over all possible values for the other boundary. The initial conditions for the outside
 5592 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 $p(X \rightsquigarrow w_{i+1:j} \mid \mathbf{w})$. This probability can be computed from the inside and outside scores,

$$p(X \rightsquigarrow w_{i+1:j} \mid \mathbf{w}) = \frac{p(X \rightsquigarrow w_{i+1:j}, \mathbf{w})}{p(\mathbf{w})} \quad [10.29]$$

$$= \frac{p(w_{i+1:j} \mid X) \times p(X, w_{1:i}, w_{j+1:M})}{p(\mathbf{w})} \quad [10.30]$$

$$= \frac{\exp(\alpha(i, j, X) + \beta(i, j, X))}{\exp \alpha(0, M, S)}. \quad [10.31]$$

5593 Marginal probabilities of individual productions can be computed similarly (see exercise
 5594 2). These marginal probabilities can be used for training a conditional random field parser,
 5595 and also for the task of unsupervised **grammar induction**, in which a PCFG is estimated
 5596 from a dataset of unlabeled text (Lari and Young, 1990; Pereira and Schabes, 1992).

5597 **10.4.4 Neural context-free grammars**

5598 Neural networks can be applied to parsing by representing each span with a dense
 5599 numerical vector (Socher et al., 2013; Durrett and Klein, 2015; Cross and Huang, 2016).⁴
 5600 For example, the anchor (i, j, k) and sentence w can be associated with a fixed-length
 5601 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),

$$\tilde{\mathbf{v}}_{(i,j,k)} = \text{FeedForward}(\mathbf{v}_{(i,j,k)}) \quad [10.33]$$

$$\psi(X \rightarrow \alpha, (i, j, k)) = \tilde{\mathbf{v}}_{(i,j,k)}^\top \Theta \mathbf{f}(X \rightarrow \alpha), \quad [10.34]$$

5602 where $\mathbf{f}(X \rightarrow \alpha)$ is a vector of features of the production, and Θ is a parameter matrix.
 5603 The matrix Θ and the parameters of the feedforward network can be learned by
 5604 backpropagating from an objective such as the margin loss or the negative conditional
 5605 log-likelihood.

5606 **10.5 Grammar refinement**

5607 The locality assumptions underlying CFG parsing depend on the granularity of the non-
 5608 terminals. For the Penn Treebank non-terminals, there are several reasons to believe that
 5609 these assumptions are too strong (Johnson, 1998):

- 5610 • The context-free assumption is too strict: for example, the probability of the production
 5611 $\text{NP} \rightarrow \text{NP PP}$ is much higher (in the PTB) if the parent of the noun phrase is a
 5612 verb phrase (indicating that the NP is a direct object) than if the parent is a sentence
 5613 (indicating that the NP is the subject of the sentence).
- 5614 • The Penn Treebank non-terminals are too coarse: there are many kinds of noun
 5615 phrases and verb phrases, and accurate parsing sometimes requires knowing the
 5616 difference. As we have already seen, when faced with prepositional phrase at-
 5617 tachment ambiguity, a weighted CFG will either always choose NP attachment (if
 5618 $\psi(\text{NP} \rightarrow \text{NP PP}) > \psi(\text{VP} \rightarrow \text{VP PP})$), or it will always choose VP attachment. To
 5619 get more nuanced behavior, more fine-grained non-terminals are needed.
- 5620 • More generally, accurate parsing requires some amount of **semantics** — understand-
 5621 ing the meaning of the text to be parsed. Consider the example *cats scratch people*

⁴Earlier work on neural constituent parsing used transition-based parsing algorithms (§ 10.6.2) rather than CKY-style chart parsing (Henderson, 2004; Titov and Henderson, 2007).

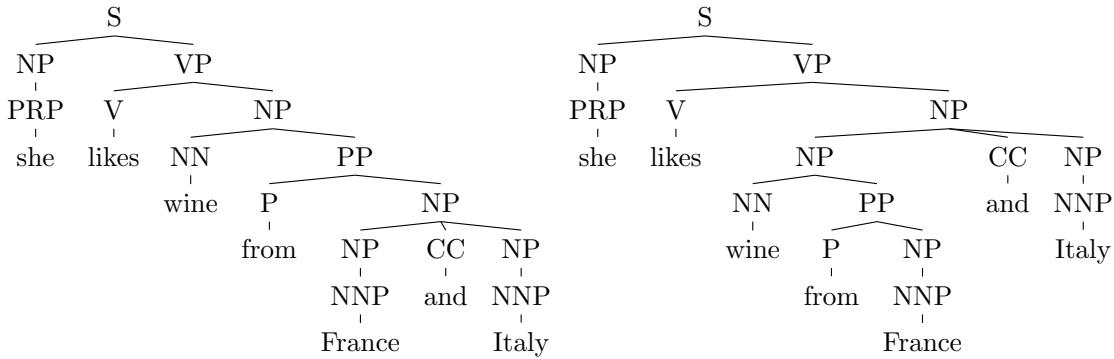


Figure 10.4: The left parse is preferable because of the conjunction of phrases headed by *France* and *Italy*, but these parses cannot be distinguished by a WCFG.

5622 *with claws*: knowledge of about cats, claws, and scratching is necessary to correctly
 5623 resolve the attachment ambiguity.

5624 An extreme example is shown in Figure 10.4. The analysis on the left is preferred
 5625 because of the conjunction of similar entities *France* and *Italy*. But given the non-terminals
 5626 shown in the analyses, there is no way to differentiate these two parses, since they include
 5627 exactly the same productions. What is needed seems to be more precise non-terminals.
 5628 One possibility would be to rethink the linguistics behind the Penn Treebank, and ask
 5629 the annotators to try again. But the original annotation effort took five years, and there
 5630 is a little appetite for another annotation effort of this scope. Researchers have therefore
 5631 turned to automated techniques.

5632 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

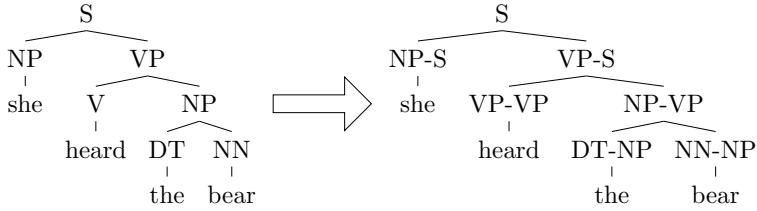


Figure 10.5: Parent annotation in a CFG derivation

of S. The observed statistics are (Johnson, 1998):

$$\Pr(\text{NP} \rightarrow \text{NP PP}) = 11\% \quad [10.35]$$

$$\Pr(\text{NP under } S \rightarrow \text{NP PP}) = 9\% \quad [10.36]$$

$$\Pr(\text{NP under } \text{VP} \rightarrow \text{NP PP}) = 23\%. \quad [10.37]$$

5633 This phenomenon can be captured by **parent annotation** (Johnson, 1998), in which each
 5634 non-terminal is augmented with the identity of its parent, as shown in Figure 10.5). This is
 5635 sometimes called **vertical Markovization**, since a Markov dependency is introduced be-
 5636 tween each node and its parent (Klein and Manning, 2003). It is analogous to moving from
 5637 a bigram to a trigram context in a hidden Markov model. In principle, parent annotation
 5638 squares the size of the set of non-terminals, which could make parsing considerably less
 5639 efficient. But in practice, the increase in the number of non-terminals that actually appear
 5640 in the data is relatively modest (Johnson, 1998).

5641 Parent annotation weakens the WCFG locality assumptions. This improves accuracy
 5642 by enabling the parser to make more fine-grained distinctions, which better capture real
 5643 linguistic phenomena. However, each production is more rare, and so careful smoothing
 5644 or regularization is required to control the variance over production scores.

5645 10.5.2 Lexicalized context-free grammars

5646 The examples in § 10.2.2 demonstrate the importance of individual words in resolving
 5647 parsing ambiguity: the preposition *on* is more likely to attach to *met*, while the preposition
 5648 *of* is more likely to attachment to *President*. But of all word pairs, which are relevant to
 5649 attachment decisions? Consider the following variants on the original examples:

- 5650 (10.3) a. We met the President of Mexico.
 5651 b. We met the first female President of Mexico.
 5652 c. They had supposedly met the President on Monday.

5653 The underlined words are the **head words** of their respective phrases: *met* heads the verb
 5654 phrase, and *President* heads the direct object noun phrase. These heads provide useful

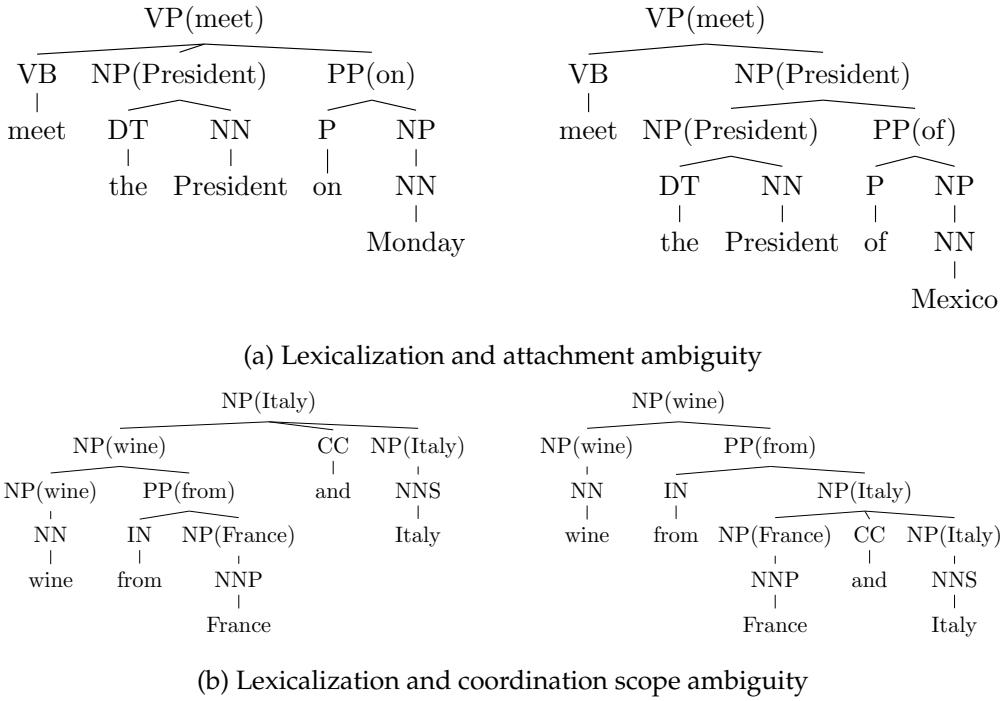


Figure 10.6: Examples of lexicalization

semantic information. But they break the context-free assumption, which states that the score for a production depends only on the parent and its immediate children, and not the substructure under each child.

The incorporation of head words into context-free parsing is known as **lexicalization**, and is implemented in rules of the form,

$$\text{NP}(President) \rightarrow \text{NP}(President) \text{ PP}(of) \quad [10.38]$$

$$\text{NP}(President) \rightarrow \text{NP}(President) \text{ PP}(on). \quad [10.39]$$

Lexicalization was a major step towards accurate PCFG parsing in the 1990s and early 2000s. It requires solving three problems: identifying the heads of all constituents in a treebank; parsing efficiently while keeping track of the heads; and estimating the scores for lexicalized productions.

| 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 (Magerman, 1995; Collins, 1997)

5662 Identifying head words

5663 The head of a constituent is the word that is the most useful for determining how that
 5664 constituent is integrated into the rest of the sentence.⁵ The head word of a constituent is
 5665 determined recursively: for any non-terminal production, the head of the left-hand side
 5666 must be the head of one of the children. The head is typically selected according to a set of
 5667 deterministic rules, sometimes called **head percolation rules**. In many cases, these rules
 5668 are straightforward: the head of a noun phrase in a $NP \rightarrow DET\ NN$ production is the head
 5669 of the noun; the head of a sentence in a $S \rightarrow NP\ VP$ production is the head of the verb
 5670 phrase.

5671 Table 10.3 shows a fragment of the head percolation rules used in many English pars-
 5672 ing systems. The meaning of the first rule is that to find the head of an S constituent, first
 5673 look for the rightmost VP child; if you don't find one, then look for the rightmost SBAR
 5674 child, and so on down the list. Verb phrases are headed by left verbs (the head of *can plan*
 5675 *on walking* is *planned*, since the modal verb *can* is tagged MD); noun phrases are headed by
 5676 the rightmost noun-like non-terminal (so the head of *the red cat* is *cat*),⁶ and prepositional
 5677 phrases are headed by the preposition (the head of *at Georgia Tech* is *at*). Some of these
 5678 rules are somewhat arbitrary — there's no particular reason why the head of *cats and dogs*
 5679 should be *dogs* — but the point here is just to get some lexical information that can support
 5680 parsing, not to make deep claims about syntax. Figure 10.6 shows the application of these
 5681 rules to two of the running examples.

5682 Parsing lexicalized context-free grammars

5683 A naïve application of lexicalization would simply increase the set of non-terminals by
 5684 taking the cross-product with the set of terminal symbols, so that the non-terminals now

⁵This is a pragmatic definition, befitting our goal of using head words to improve parsing; for a more formal definition, see (Bender, 2013, chapter 7).

⁶The noun phrase non-terminal is sometimes treated as a special case. Collins (1997) uses a heuristic that looks for the rightmost child which is a noun-like part-of-speech (e.g., NN, NNP), a possessive marker, or a superlative adjective (e.g., *the greatest*). If no such child is found, the heuristic then looks for the *leftmost* NP. If there is no child with tag NP, the heuristic then applies another priority list, this time from right to left.

5685 include symbols like $\text{NP}(\text{President})$ and $\text{VP}(\text{meet})$. Under this approach, the CKY parsing
 5686 algorithm could be applied directly to the lexicalized production rules. However, the
 5687 complexity would be cubic in the size of the vocabulary of terminal symbols, which would
 5688 clearly be intractable.

Another approach is to augment the CKY table with an additional index, keeping track of the head of each constituent. The cell $t[i, j, h, X]$ stores the score of the best derivation in which non-terminal X spans $w_{i+1:j}$ with head word h , where $i < h \leq j$. To compute such a table recursively, we must consider the possibility that each phrase gets its head from either its left or right child. The scores of the best derivations in which the head comes from the left and right child are denoted t_ℓ and t_r respectively, leading to the following recurrence:

$$t_\ell[i, j, h, X] = \max_{(X \rightarrow YZ)} \max_{k > h} \max_{k < h' \leq j} t[i, k, h, Y] + t[k, j, h', Z] + \psi(X(h) \rightarrow Y(h)Z(h')) \quad [10.40]$$

$$t_r[i, j, h, X] = \max_{(X \rightarrow YZ)} \max_{k < h} \max_{i < h' \leq k} t[i, k, h', Y] + t[k, j, h, Z] + (\psi(X(h) \rightarrow Y(h')Z(h))) \quad [10.41]$$

$$t[i, j, h, X] = \max(t_\ell[i, j, h, X], t_r[i, j, h, X]). \quad [10.42]$$

5689 To compute t_ℓ , we maximize over all split points $k > h$, since the head word must be in
 5690 the left child. We then maximize again over possible head words h' for the right child. An
 5691 analogous computation is performed for t_r . The size of the table is now $\mathcal{O}(M^3N)$, where
 5692 M is the length of the input and N is the number of non-terminals. Furthermore, each
 5693 cell is computed by performing $\mathcal{O}(M^2)$ operations, since we maximize over both the split
 5694 point k and the head h' . The time complexity of the algorithm is therefore $\mathcal{O}(RM^5N)$,
 5695 where R is the number of rules in the grammar. Fortunately, more efficient solutions are
 5696 possible. In general, the complexity of parsing can be reduced to $\mathcal{O}(M^4)$ in the length of
 5697 the input; for a broad class of lexicalized CFGs, the complexity can be made cubic in the
 5698 length of the input, just as in unlexicalized CFGs (Eisner, 2000).

5699 Estimating lexicalized context-free grammars

5700 The final problem for lexicalized parsing is how to estimate weights for lexicalized pro-
 5701 ductions $X(i) \rightarrow Y(j) Z(k)$. These productions are said to be bilexical, because they
 5702 involve scores over pairs of words: in the example *meet the President of Mexico*, we hope
 5703 to choose the correct attachment point by modeling the bilexical affinities of (*meet, of*) and
 5704 (*President, of*). The number of such word pairs is quadratic in the size of the vocabulary,
 5705 making it difficult to estimate the weights of lexicalized production rules directly from
 5706 data. This is especially true for probabilistic context-free grammars, in which the weights
 5707 are obtained from smoothed relative frequency. In a treebank with a million tokens, a

5708 vanishingly small fraction of the possible lexicalized productions will be observed more
 5709 than once.⁷ The Charniak (1997) and Collins (1997) parsers therefore focus on approximating
 5710 the probabilities of lexicalized productions, using various smoothing techniques
 5711 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:

$$\begin{aligned} f(\text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)) = & \{\text{NP}(*) \rightarrow \text{NP}(*) \text{ PP}(*), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(*), \\ & \text{NP}(*) \rightarrow \text{NP}(*) \text{ PP}(of), \\ & \text{NP}(\text{President}) \rightarrow \text{NP}(\text{President}) \text{ PP}(of)\} \end{aligned}$$

5712 The first feature scores the unlexicalized production $\text{NP} \rightarrow \text{NP PP}$; the next two features
 5713 lexicalize only one element of the production, thereby scoring the appropriateness of NP
 5714 attachment for the individual words *President* and *of*; the final feature scores the specific
 5715 blexical affinity of *President* and *of*. For blexical pairs that are encountered frequently in
 5716 the treebank, this blexical feature can play an important role in parsing; for pairs that are
 5717 absent or rare, regularization will drive its weight to zero, forcing the parser to rely on the
 5718 more coarse-grained features.

5719 In chapter 14, we will encounter techniques for clustering words based on their **distribu-**
 5720 **tional** properties — the contexts in which they appear. Such a clustering would group
 5721 rare and common words, such as *whale*, *shark*, *beluga*, *Leviathan*. Word clusters can be used
 5722 as features in discriminative lexicalized parsing, striking a middle ground between full
 5723 lexicalization and non-terminals (Finkel et al., 2008). In this way, labeled examples con-
 5724 taining relatively common words like *whale* can help to improve parsing for rare words
 5725 like *beluga*, as long as those two words are clustered together.

5726 10.5.3 *Refinement grammars

5727 Lexicalization improves on context-free parsing by adding detailed information in the
 5728 form of lexical heads. However, estimating the scores of lexicalized productions is dif-
 5729 ficult. Klein and Manning (2003) argue that the right level of linguistic detail is some-
 5730 where between treebank categories and individual words. Some parts-of-speech and non-
 5731 terminals are truly substitutable: for example, *cat*/N and *dog*/N. But others are not: for
 5732 example, the preposition *of* exclusively attaches to nouns, while the preposition *as* is more

⁷The real situation is even more difficult, because non-binary context-free grammars can involve trilexical or higher-order dependencies, between the head of the constituent and multiple of its children (Carreras et al., 2008).

likely to modify verb phrases. Klein and Manning (2003) obtained a 2% improvement in *F*-MEASURE on a parent-annotated PCFG parser by making a single change: splitting the preposition category into six subtypes. They propose a series of linguistically-motivated refinements to the Penn Treebank annotations, which in total yielded a 40% error reduction.

Non-terminal refinement process can be automated by treating the refined categories as **latent variables**. For example, we might split the noun phrase non-terminal into NP1, NP2, NP3, . . . , without defining in advance what each refined non-terminal corresponds to. This can be treated as partially supervised learning, similar to the multi-component document classification model described in § 5.2.3. A latent variable PCFG can be estimated by expectation maximization (Matsuzaki et al., 2005):⁸

- In the E-step, estimate a marginal distribution q over the refinement type of each non-terminal in each derivation. These marginals are constrained by the original annotation: an NP can be reannotated as NP4, but not as VP3. Marginal probabilities over refined productions can be computed from the **inside-outside algorithm**, as described in § 10.4.3, where the E-step enforces the constraints imposed by the original annotations.
- In the M-step, recompute the parameters of the grammar, by summing over the probabilities of anchored productions that were computed in the E-step:

$$E[\text{count}(X \rightarrow Y Z)] = \sum_{i=0}^M \sum_{j=i}^M \sum_{k=i}^j p(X \rightarrow Y Z, (i, j, k) \mid \mathbf{w}). \quad [10.43]$$

As usual, this process can be iterated to convergence. To determine the number of refinement types for each tag, Petrov et al. (2006) apply a split-merge heuristic; Liang et al. (2007) and Finkel et al. (2007) apply **Bayesian nonparametrics** (Cohen, 2016).

Some examples of refined non-terminals are shown in Table 10.4. The proper nouns differentiate months, first names, middle initials, last names, first names of places, and second names of places; each of these will tend to appear in different parts of grammatical productions. The personal pronouns differentiate grammatical role, with PRP-0 appearing in subject position at the beginning of the sentence (note the capitalization), PRP-1 appearing in subject position but not at the beginning of the sentence, and PRP-2 appearing in object position.

10.6 Beyond context-free parsing

In the context-free setting, the score for a parse is a combination of the scores of individual productions. As we have seen, these models can be improved by using finer-grained non-

⁸Spectral learning, described in § 5.5.2, has also been applied to refinement grammars (Cohen et al., 2014).

| 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).

5765 terminals, via parent-annotation, lexicalization, and automated refinement. However, the
 5766 inherent limitations to the expressiveness of context-free parsing motivate the consider-
 5767 ation of other search strategies. These strategies abandon the optimality guaranteed by
 5768 bottom-up parsing, in exchange for the freedom to consider arbitrary properties of the
 5769 proposed parses.

5770 10.6.1 Reranking

5771 A simple way to relax the restrictions of context-free parsing is to perform a two-stage pro-
 5772 cess, in which a context-free parser generates a k -best list of candidates, and a **reranker**
 5773 then selects the best parse from this list (Charniak and Johnson, 2005; Collins and Koo,
 5774 2005). The reranker can be trained from an objective that is similar to multi-class classi-
 5775 fication: the goal is to learn weights that assign a high score to the reference parse, or to
 5776 the parse on the k -best list that has the lowest error. In either case, the reranker need only
 5777 evaluate the K best parses, and so no context-free assumptions are necessary. This opens
 5778 the door to more expressive scoring functions:

- 5779 • It is possible to incorporate arbitrary non-local features, such as the structural par-
 5780 allelism and right-branching orientation of the parse (Charniak and Johnson, 2005).
- 5781 • Reranking enables the use of **recursive neural networks**, in which each constituent
 5782 span $w_{i+1:j}$ receives a vector $u_{i,j}$ which is computed from the vector representa-
 5783 tions of its children, using a composition function that is linked to the production

5784 rule (Socher et al., 2013), e.g.,

$$\mathbf{u}_{i,j} = f\left(\Theta_{X \rightarrow Y \mid Z} \begin{bmatrix} \mathbf{u}_{i,k} \\ \mathbf{u}_{k,j} \end{bmatrix}\right) \quad [10.44]$$

5785 The overall score of the parse can then be computed from the final vector, $\Psi(\tau) =$
 5786 $\theta \mathbf{u}_{0,M}$.

5787 Reranking can yield substantial improvements in accuracy. The main limitation is that it
 5788 can only find the best parse among the K -best offered by the generator, so it is inherently
 5789 limited by the ability of the bottom-up parser to find high-quality candidates.

5790 10.6.2 Transition-based parsing

5791 Structure prediction can be viewed as a form of search. An alternative to bottom-up pars-
 5792 ing is to read the input from left-to-right, gradually building up a parse structure through
 5793 a series of **transitions**. Transition-based parsing is described in more detail in the next
 5794 chapter, in the context of dependency parsing. However, it can also be applied to CFG
 5795 parsing, as briefly described here.

5796 For any context-free grammar, there is an equivalent **pushdown automaton**, a model
 5797 of computation that accepts exactly those strings that can be derived from the grammar.
 5798 This computational model consumes the input from left to right, while pushing and pop-
 5799 ping elements on a stack. This architecture provides a natural transition-based parsing
 5800 framework for context-free grammars, known as **shift-reduce parsing**.

5801 Shift-reduce parsing is a type of transition-based parsing, in which the parser can take
 5802 the following actions:

- 5803 • *shift* the next terminal symbol onto the stack;
- 5804 • *unary-reduce* the top item on the stack, using a unary production rule in the gram-
 5805 mar;
- 5806 • *binary-reduce* the top two items onto the stack, using a binary production rule in the
 5807 grammar.

5808 The set of available actions is constrained by the situation: the parser can only shift if
 5809 there are remaining terminal symbols in the input, and it can only reduce if an applicable
 5810 production rule exists in the grammar. If the parser arrives at a state where the input
 5811 has been completely consumed, and the stack contains only the element S, then the input
 5812 is accepted. If the parser arrives at a non-accepting state where there are no possible
 5813 actions, the input is rejected. A parse error occurs if there is some action sequence that
 5814 would accept an input, but the parser does not find it.

5815 **Example** Consider the input *we eat sushi* and the grammar in Table 10.1. The input can
 5816 be parsed through the following sequence of actions:

- 5817 1. **Shift** the first token *we* onto the stack.
- 5818 2. **Reduce** the top item on the stack to NP, using the production $NP \rightarrow we$.
- 5819 3. **Shift** the next token *eat* onto the stack, and **reduce** it to V with the production $V \rightarrow$
 5820 *eat*.
- 5821 4. **Shift** the final token *sushi* onto the stack, and **reduce** it to NP. The input has been
 5822 completely consumed, and the stack contains [NP, V, NP].
- 5823 5. **Reduce** the top two items using the production $VP \rightarrow V NP$. The stack now con-
 5824 tains [VP, NP].
- 5825 6. **Reduce** the top two items using the production $S \rightarrow NP VP$. The stack now contains
 5826 [S]. Since the input is empty, this is an accepting state.

5827 One thing to notice from this example is that the number of shift actions is equal to the
 5828 length of the input. The number of reduce actions is equal to the number of non-terminals
 5829 in the analysis, which grows linearly in the length of the input. Thus, the overall time
 5830 complexity of shift-reduce parsing is linear in the length of the input (assuming the com-
 5831 plexity of each individual classification decision is constant in the length of the input).
 5832 This is far better than the cubic time complexity required by CKY parsing.

5833 **Transition-based parsing as inference** In general, it is not possible to guarantee that
 5834 a transition-based parser will find the optimal parse, $\text{argmax}_\tau \Psi(\tau; \mathbf{w})$, even under the
 5835 usual CFG independence assumptions. We could assign a score to each anchored parsing
 5836 action in each context, with $\psi(a, c)$ indicating the score of performing action a in context c .
 5837 One might imagine that transition-based parsing could efficiently find the derivation that
 5838 maximizes the sum of such scores. But this too would require backtracking and searching
 5839 over an exponentially large number of possible action sequences: if a bad decision is
 5840 made at the beginning of the derivation, then it may be impossible to recover the optimal
 5841 action sequence without backtracking to that early mistake. This is known as a **search**
 5842 **error**. Transition-based parsers can incorporate arbitrary features, without the restrictive
 5843 independence assumptions required by chart parsing; search errors are the price that must
 5844 be paid for this flexibility.

5845 **Learning transition-based parsing** Transition-based parsing can be combined with ma-
 5846 chine learning by training a classifier to select the correct action in each situation. This
 5847 classifier is free to choose any feature of the input, the state of the parser, and the parse
 5848 history. However, there is no optimality guarantee: the parser may choose a suboptimal
 5849 parse, due to a mistake at the beginning of the analysis. Nonetheless, some of the strongest

5850 CFG parsers are based on the shift-reduce architecture, rather than CKY. A recent genera-
 5851 tion of models links shift-reduce parsing with recurrent neural networks, updating a
 5852 hidden state vector while consuming the input (e.g., Cross and Huang, 2016; Dyer et al.,
 5853 2016). Learning algorithms for transition-based parsing are discussed in more detail in
 5854 § 11.3.

5855 Exercises

5856 1. Design a grammar that handles English subject-verb agreement. Specifically, your
 5857 grammar should handle the examples below correctly:

5858 (10.4) a. She sings.

5859 b. We sing.

5860 (10.5) a. *She sing.

5861 b. *We sings.

5862 2. Extend your grammar from the previous problem to include the auxiliary verb *can*,
 5863 so that the following cases are handled:

5864 (10.6) a. She can sing.

5865 b. We can sing.

5866 (10.7) a. *She can sings.

5867 b. *We can sings.

5868 3. French requires subjects and verbs to agree in person and number, and it requires
 5869 determiners and nouns to agree in gender and number. Verbs and their objects need
 5870 not agree. Assuming that French has two genders (feminine and masculine), three
 5871 persons (first [*me*], second [*you*], third [*her*]), and two numbers (singular and plural),
 5872 how many productions are required to extend the following simple grammar to
 5873 handle agreement?

| | | |
|----|---------------|--------------------|
| S | \rightarrow | NP VP |
| VP | \rightarrow | V V NP V NP NP |
| NP | \rightarrow | DET NN |

5875 4. Consider the grammar:

| | |
|------|---|
| 5876 | S → NP VP VP → V NP NP → JJ NP NP → <i>fish</i> (the animal) V → <i>fish</i> (the action of fishing) JJ → <i>fish</i> (a modifier, as in <i>fish sauce</i> or <i>fish stew</i>) |
|------|---|

5877 Apply the CKY algorithm and identify all possible parses for the sentence *fish fish*
 5878 *fish fish*.

- 5879 5. Choose one of the possible parses for the previous problem, and show how it can be
 5880 derived by a series of shift-reduce actions.
- 5881 6. To handle VP coordination, a grammar includes the production $VP \rightarrow VP\ CC\ VP$.
 5882 To handle adverbs, it also includes the production $VP \rightarrow VP\ ADV$. Assume all verbs
 5883 are generated from a sequence of unary productions, e.g., $VP \rightarrow V \rightarrow eat$.
- 5884 a) Show how to binarize the production $VP \rightarrow VP\ CC\ VP$.
- 5885 b) Use your binarized grammar to parse the sentence *They eat and drink together*,
 5886 treating *together* as an adverb.
- 5887 c) Prove that a weighted CFG cannot distinguish the two possible derivations of
 5888 this sentence. Your explanation should focus on the productions in the original,
 5889 non-binary grammar.
- 5890 d) Explain what condition must hold for a parent-annotated WCFG to prefer the
 5891 derivation in which *together* modifies the coordination *eat and drink*.

7. Consider the following PCFG:

$$p(X \rightarrow X\ X) = \frac{1}{2} \quad [10.45]$$

$$p(X \rightarrow Y) = \frac{1}{2} \quad [10.46]$$

$$p(Y \rightarrow \sigma) = \frac{1}{|\Sigma|}, \forall \sigma \in \Sigma \quad [10.47]$$

- 5892 a) Compute the probability $p(\hat{\tau})$ of the maximum probability parse for a string
 5893 $w \in \Sigma^M$.
- 5894 b) Compute the conditional probability $p(\hat{\tau} | w)$.
- 5895 8. Context-free grammars can be used to parse the internal structure of words. Us-
 5896 ing the weighted CKY algorithm and the following weighted context-free grammar,
 5897 identify the best parse for the sequence of morphological segments *in+flame+able*.

| | | | | |
|------|---------|---|-----------|----|
| | S | → | V | 0 |
| | S | → | N | 0 |
| | S | → | J | 0 |
| | V | → | VPref N | -1 |
| 5898 | J | → | N JSuff | 1 |
| | J | → | V JSuff | 0 |
| | J | → | NegPref J | 1 |
| | VPref | → | in+ | 2 |
| | NegPref | → | in+ | 1 |
| | N | → | flame | 0 |
| | JSuff | → | +able | 0 |

- 5899 9. Use the inside and outside scores to compute the marginal probability $p(X_{i+1:j} \rightarrow Y_{i+1:k} Z_{k+1:j} \mid \mathbf{w})$,
 5900 indicating that Y spans $\mathbf{w}_{i+1:k}$, Z spans $\mathbf{w}_{k+1:j}$, and X is the parent of Y and Z , spanning $\mathbf{w}_{i+1:j}$.
 5901
- 5902 10. Suppose that the potentials $\Psi(X \rightarrow \alpha)$ are log-probabilities, so that $\sum_{\alpha} \exp \Psi(X \rightarrow \alpha) = 1$
 5903 for all X . Verify that the semiring inside recurrence from Equation 10.26 generates
 5904 the log-probability $\log p(\mathbf{w}) = \log \sum_{\tau: \text{yield}(\tau) = \mathbf{w}} p(\tau)$.

5905 Chapter 11

5906 Dependency parsing

5907 The previous chapter discussed algorithms for analyzing sentences in terms of nested con-
5908 stituents, such as noun phrases and verb phrases. However, many of the key sources of
5909 ambiguity in phrase-structure analysis relate to questions of **attachment**: where to attach a
5910 prepositional phrase or complement clause, how to scope a coordinating conjunction, and
5911 so on. These attachment decisions can be represented with a more lightweight structure:
5912 a directed graph over the words in the sentence, known as a **dependency parse**. Syntac-
5913 tic annotation has shifted its focus to such dependency structures: at the time of this
5914 writing, the **Universal Dependencies** project offers more than 100 dependency treebanks
5915 for more than 60 languages.¹ This chapter will describe the linguistic ideas underlying
5916 dependency grammar, and then discuss exact and transition-based parsing algorithms.
5917 The chapter will also discuss recent research on **learning to search** in transition-based
5918 structure prediction.

5919 11.1 Dependency grammar

5920 While **dependency grammar** has a rich history of its own (Tesnière, 1966; Kübler et al.,
5921 2009), it can be motivated by extension from the lexicalized context-free grammars that
5922 we encountered in previous chapter (§ 10.5.2). Recall that lexicalization augments each
5923 non-terminal with a **head word**. The head of a constituent is identified recursively, using
5924 a set of **head rules**, as shown in Table 10.3. An example of a lexicalized context-free parse
5925 is shown in Figure 11.1a. In this sentence, the head of the S constituent is the main verb,
5926 *scratch*; this non-terminal then produces the noun phrase *the cats*, whose head word is
5927 *cats*, and from which we finally derive the word *the*. Thus, the word *scratch* occupies the
5928 central position for the sentence, with the word *cats* playing a supporting role. In turn, *cats*

¹universaldependencies.org

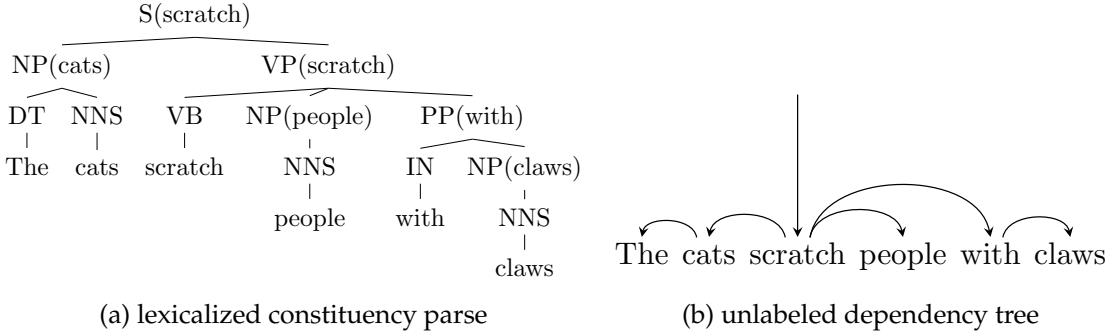


Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent. (This example is based on the lexicalization rules from § 10.5.2, which make the preposition the head of a prepositional phrase. In the more contemporary Universal Dependencies annotations, the head of *with claws* would be *claws*, so there would be an edge *scratch* → *claws*.)

occupies the central position for the noun phrase, with the word *the* playing a supporting role.

The relationships between words in a sentence can be formalized in a directed graph, based on the lexicalized phrase-structure parse: create an edge (i, j) iff word i is the head of a phrase whose child is a phrase headed by word j . Thus, in our example, we would have *scratch* → *cats* and *cats* → *the*. We would not have the edge *scratch* → *the*, because although $S(\text{scratch})$ dominates $\text{DET}(\text{the})$ in the phrase-structure parse tree, it is not its immediate parent. These edges describe **syntactic dependencies**, a blexical relationship between a **head** and a **dependent**, which is at the heart of dependency grammar.

Continuing to build out this **dependency graph**, we will eventually reach every word in the sentence, as shown in Figure 11.1b. In this graph — and in all graphs constructed in this way — every word has exactly one incoming edge, except for the root word, which is indicated by a special incoming arrow from above. Furthermore, the graph is *weakly connected*: if the directed edges were replaced with undirected edges, there would be a path between all pairs of nodes. From these properties, it can be shown that there are no cycles in the graph (or else at least one node would have to have more than one incoming edge), and therefore, the graph is a tree. Because the graph includes all vertices, it is a **spanning tree**.

11.1.1 Heads and dependents

A dependency edge implies an asymmetric syntactic relationship between the head and dependent words, sometimes called **modifiers**. For a pair like *the cats* or *cats scratch*, how

5950 do we decide which is the head? Here are some possible criteria:

- 5951 • The head sets the syntactic category of the construction: for example, nouns are the
5952 heads of noun phrases, and verbs are the heads of verb phrases.
- 5953 • The modifier may be optional while the head is mandatory: for example, in the
5954 sentence *cats scratch people with claws*, the subtrees *cats scratch* and *cats scratch people*
5955 are grammatical sentences, but *with claws* is not.
- 5956 • The head determines the morphological form of the modifier: for example, in lan-
5957 guages that require gender agreement, the gender of the noun determines the gen-
5958 der of the adjectives and determiners.
- 5959 • Edges should first connect content words, and then connect function words.

5960 These guidelines are not universally accepted, and they sometimes conflict. The Uni-
5961 versal Dependencies (UD) project has attempted to identify a set of principles that can be
5962 applied to dozens of different languages (Nivre et al., 2016).² These guidelines are based
5963 on the universal part-of-speech tags from chapter 8. They differ somewhat from the head
5964 rules described in § 10.5.2: for example, on the principle that dependencies should relate
5965 content words, the prepositional phrase *with claws* would be headed by *claws*, resulting in
5966 an edge *scratch* → *claws*, and another edge *claws* → *with*.

5967 One objection to dependency grammar is that not all syntactic relations are asymmet-
5968 ric. One such relation is coordination (Popel et al., 2013): in the sentence, *Abigail and Max*
5969 *like kimchi* (Figure 11.2), which word is the head of the coordinated noun phrase *Abigail*
5970 and *Max*? Choosing either *Abigail* or *Max* seems arbitrary; fairness argues for making *and*
5971 the head, but this seems like the least important word in the noun phrase, and selecting
5972 it would violate the principle of linking content words first. The Universal Dependencies
5973 annotation system arbitrarily chooses the left-most item as the head — in this case, *Abigail*
5974 — and includes edges from this head to both *Max* and the coordinating conjunction *and*.
5975 These edges are distinguished by the labels CONJ (for the thing begin conjoined) and CC
5976 (for the coordinating conjunction). The labeling system is discussed next.

5977 11.1.2 Labeled dependencies

5978 Edges may be **labeled** to indicate the nature of the syntactic relation that holds between
5979 the two elements. For example, in Figure 11.2, the label NSUBJ on the edge from *like* to
5980 *Abigail* indicates that the subtree headed by *Abigail* is the noun subject of the verb *like*;
5981 similarly, the label OBJ on the edge from *like* to *kimchi* indicates that the subtree headed by

²The latest and most specific guidelines are available at universaldependencies.org/guidelines.html

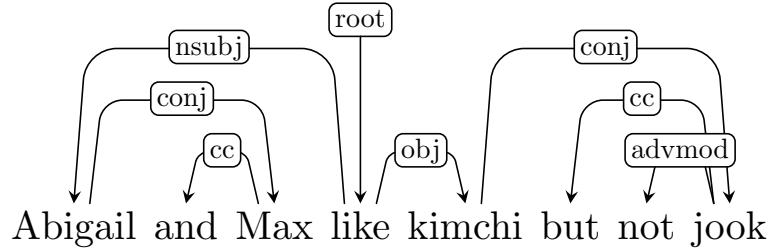


Figure 11.2: In the Universal Dependencies annotation system, the left-most item of a coordination is the head.

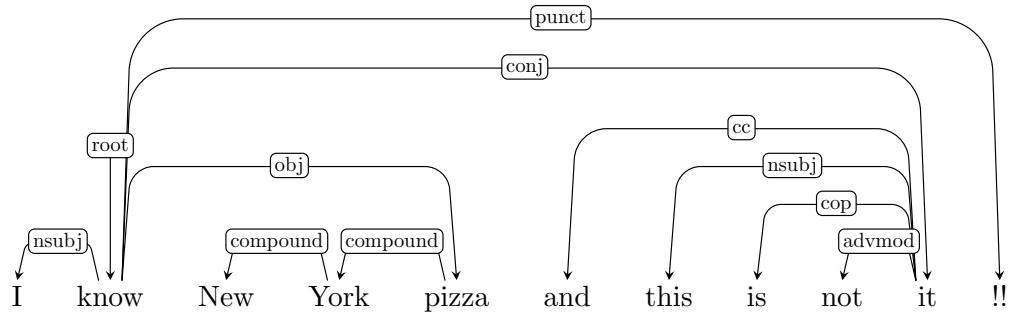


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

5982 *kimchi* is the object.³ The negation *not* is treated as an adverbial modifier (ADVMOD) on
5983 the noun *jook*.

5984 A slightly more complex example is shown in Figure 11.3. The multiword expression
5985 *New York pizza* is treated as a “flat” unit of text, with the elements linked by the COM-
5986 POUND relation. The sentence includes two clauses that are conjoined in the same way
5987 that noun phrases are conjoined in Figure 11.2. The second clause contains a **copula** verb
5988 (see § 8.1.1). For such clauses, we treat the “object” of the verb as the root — in this case,
5989 *it* — and label the verb as a dependent, with the COP relation. This example also shows
5990 how punctuations are treated, with label PUNCT.

5991 11.1.3 Dependency subtrees and constituents

5992 Dependency trees hide information that would be present in a CFG parse. Often what
5993 is hidden is in fact irrelevant: for example, Figure 11.4 shows three different ways of

³Earlier work distinguished direct and indirect objects (De Marneffe and Manning, 2008), but this has been dropped in version 2.0 of the Universal Dependencies annotation system.

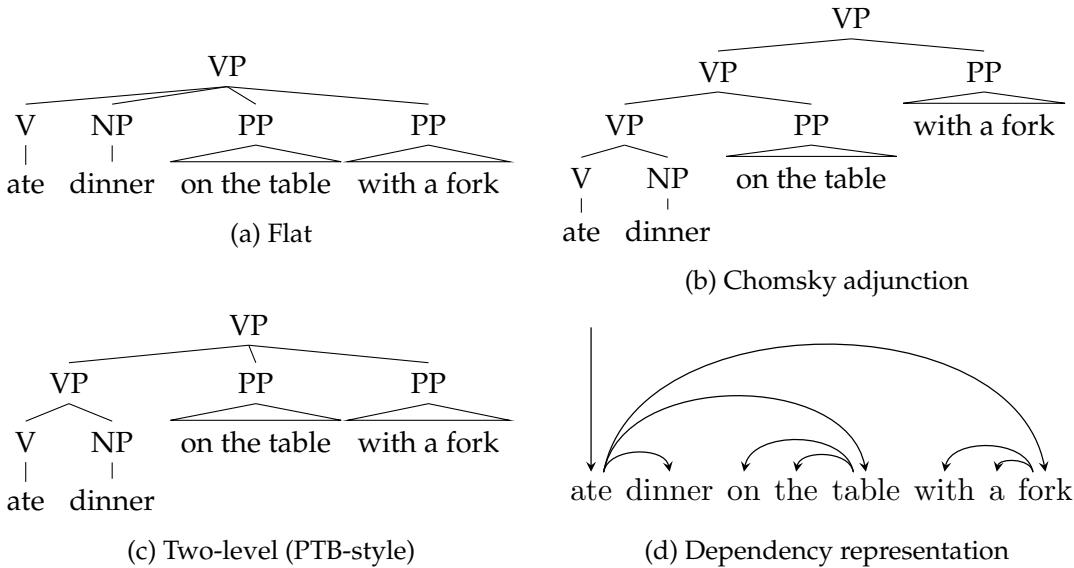


Figure 11.4: The three different CFG analyses of this verb phrase all correspond to a single dependency structure.

representing prepositional phrase adjuncts to the verb *ate*. Because there is apparently no meaningful difference between these analyses, the Penn Treebank decides by convention to use the two-level representation (see Johnson, 1998, for a discussion). As shown in Figure 11.4d, these three cases all look the same in a dependency parse.

But dependency grammar imposes its own set of annotation decisions, such as the identification of the head of a coordination (§ 11.1.1); without lexicalization, context-free grammar does not require either element in a coordination to be privileged in this way. Dependency parses can be disappointingly flat: for example, in the sentence *Yesterday, Abigail was reluctantly giving Max kimchi*, the root *giving* is the head of every dependency! The constituent parse arguably offers a more useful structural analysis for such cases.

Projectivity Thus far, we have defined dependency trees as spanning trees over a graph in which each word is a vertex. As we have seen, one way to construct such trees is by connecting the heads in a lexicalized constituent parse. However, there are spanning trees that cannot be constructed in this way. Syntactic constituents are *contiguous spans*. In a spanning tree constructed from a lexicalized constituent parse, the head h of any constituent that spans the nodes from i to j must have a path to every node in this span. This property is known as **projectivity**, and projective dependency parses are a restricted class of spanning trees. Informally, projectivity means that “crossing edges” are prohibited. The formal definition follows:

| | % non-projective edges | % non-projective sentences |
|---------|------------------------|----------------------------|
| Czech | 1.86% | 22.42% |
| English | 0.39% | 7.63% |
| German | 2.33% | 28.19% |

Table 11.1: Frequency of non-projective dependencies in three languages (Kuhlmann and Nivre, 2010)

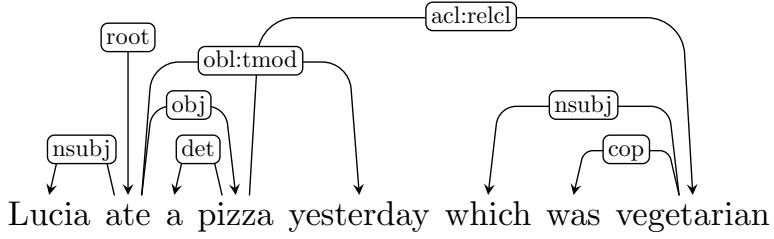


Figure 11.5: An example of a non-projective dependency parse. The “crossing edge” arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

6013 **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.*

6015 Figure 11.5 gives an example of a non-projective dependency graph in English. This
6016 dependency graph does not correspond to any constituent parse. As shown in Table 11.1,
6017 non-projectivity is more common in languages such as Czech and German. Even though
6018 relatively few dependencies are non-projective in these languages, many sentences have
6019 at least one such dependency. As we will soon see, projectivity has important algorithmic
6020 consequences.

6021 11.2 Graph-based dependency parsing

6022 Let $\mathbf{y} = \{i \xrightarrow{r} j\}$ represent a dependency graph, in which each edge is a relation r from
6023 head word $i \in \{1, 2, \dots, M, \text{ROOT}\}$ to modifier $j \in \{1, 2, \dots, M\}$. The special node ROOT
6024 indicates the root of the graph, and M is the length of the input $|\mathbf{w}|$. Given a scoring
6025 function $\Psi(\mathbf{y}, \mathbf{w}; \theta)$, the optimal parse is,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{w})}{\operatorname{argmax}} \Psi(\mathbf{y}, \mathbf{w}; \theta), \quad [11.1]$$

6026 where $\mathcal{Y}(\mathbf{w})$ is the set of valid dependency parses on the input \mathbf{w} . As usual, the number
6027 of possible labels $|\mathcal{Y}(\mathbf{w})|$ is exponential in the length of the input (Wu and Chao, 2004).

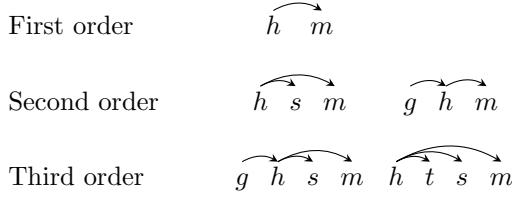


Figure 11.6: Feature templates for higher-order dependency parsing

6028 Algorithms that search over this space of possible graphs are known as **graph-based de-**
 6029 **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 $i \rightarrow j$:

$$\Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}). \quad [11.2]$$

6030 Dependency parsers that operate under this assumption are known as **arc-factored**, since
 6031 the score of a graph is the product of the scores of all arcs.

Higher-order dependency parsing The arc-factored decomposition can be relaxed to allow higher-order dependencies. In **second-order dependency parsing**, the scoring function may include grandparents and siblings, as shown by the templates in Figure 11.6. The scoring function is,

$$\begin{aligned} \Psi(\mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) = & \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi_{\text{parent}}(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \\ & + \sum_{k \xrightarrow{r'} i \in \mathbf{y}} \psi_{\text{grandparent}}(i \xrightarrow{r} j, k, r', \mathbf{w}; \boldsymbol{\theta}) \\ & + \sum_{\substack{i \xrightarrow{r'} s \in \mathbf{y} \\ s \neq j}} \psi_{\text{ sibling}}(i \xrightarrow{r} j, s, r', \mathbf{w}; \boldsymbol{\theta}). \end{aligned} \quad [11.3]$$

6032 The top line scores computes a scoring function that includes the grandparent k ; the
 6033 bottom line computes a scoring function for each sibling s . For projective dependency
 6034 graphs, there are efficient algorithms for second-order and third-order dependency pars-
 6035 ing (Eisner, 1996; McDonald and Pereira, 2006; Koo and Collins, 2010); for non-projective
 6036 dependency graphs, second-order dependency parsing is NP-hard (McDonald and Pereira,
 6037 2006). The specific algorithms are discussed in the next section.

6038 **11.2.1 Graph-based parsing algorithms**

6039 The distinction between projective and non-projective dependency trees (§ 11.1.3) plays
 6040 a key role in the choice of algorithms. Because projective dependency trees are closely
 6041 related to (and can be derived from) lexicalized constituent trees, lexicalized parsing al-
 6042 gorithms can be applied directly. For the more general problem of parsing to arbitrary
 6043 spanning trees, a different class of algorithms is required. In both cases, arc-factored de-
 6044 pendency parsing relies on precomputing the scores $\psi(i \xrightarrow{r} j, w; \theta)$ for each potential
 6045 edge. There are $\mathcal{O}(M^2 R)$ such scores, where M is the length of the input and R is the
 6046 number of dependency relation types, and this is a lower bound on the time and space
 6047 complexity of any exact algorithm for arc-factored dependency parsing.

6048 **Projective dependency parsing**

6049 Any lexicalized constituency tree can be converted into a projective dependency tree by
 6050 creating arcs between the heads of constituents and their parents, so any algorithm for
 6051 lexicalized constituent parsing can be converted into an algorithm for projective depen-
 6052 dency parsing, by converting arc scores into scores for lexicalized productions. As noted
 6053 in § 10.5.2, there are cubic time algorithms for lexicalized constituent parsing, which are
 6054 extensions of the CKY algorithm. Therefore, arc-factored projective dependency parsing
 6055 can be performed in cubic time in the length of the input.

6056 Second-order projective dependency parsing can also be performed in cubic time, with
 6057 minimal modifications to the lexicalized parsing algorithm (Eisner, 1996). It is possible to
 6058 go even further, to **third-order dependency parsing**, in which the scoring function may
 6059 consider great-grandparents, grand-siblings, and “tri-siblings”, as shown in Figure 11.6.
 6060 Third-order dependency parsing can be performed in $\mathcal{O}(M^4)$ time, which can be made
 6061 practical through the use of pruning to eliminate unlikely edges (Koo and Collins, 2010).

6062 **Non-projective dependency parsing**

6063 In non-projective dependency parsing, the goal is to identify the highest-scoring span-
 6064 ning tree over the words in the sentence. The arc-factored assumption ensures that the
 6065 score for each spanning tree will be computed as a sum over scores for the edges, which
 6066 are precomputed. Based on these scores, we build a weighted connected graph. Arc-
 6067 factored non-projective dependency parsing is then equivalent to finding the spanning
 6068 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-**
 6069 **Liu-Edmonds algorithm** (Chu and Liu, 1965; Edmonds, 1967) computes this **maximum**
 6070 **directed spanning tree** efficiently. It does this by first identifying the best incoming edge
 6071 $i \xrightarrow{r} j$ for each vertex j . If the resulting graph does not contain cycles, it is the maxi-
 6072 mum spanning tree. If there is a cycle, it is collapsed into a super-vertex, whose incoming
 6073 and outgoing edges are based on the edges to the vertices in the cycle. The algorithm is

6074 then applied recursively to the resulting graph, and process repeats until a graph without
 6075 cycles is obtained.

6076 The time complexity of identifying the best incoming edge for each vertex is $\mathcal{O}(M^2R)$,
 6077 where M is the length of the input and R is the number of relations; in the worst case, the
 6078 number of cycles is $\mathcal{O}(M)$. Therefore, the complexity of the Chu-Liu-Edmonds algorithm
 6079 is $\mathcal{O}(M^3R)$. This complexity can be reduced to $\mathcal{O}(M^2N)$ by storing the edge scores in a
 6080 Fibonacci heap (Gabow et al., 1986). For more detail on graph-based parsing algorithms,
 6081 see Eisner (1997) and Kübler et al. (2009).

6082 **Higher-order non-projective dependency parsing** Given the tractability of higher-order
 6083 projective dependency parsing, you may be surprised to learn that non-projective second-
 6084 order dependency parsing is NP-Hard. This can be proved by reduction from the vertex
 6085 cover problem (Neuhaus and Bröker, 1997). A heuristic solution is to do projective pars-
 6086 ing first, and then post-process the projective dependency parse to add non-projective
 6087 edges (Nivre and Nilsson, 2005). More recent work has applied techniques for approxi-
 6088 mate inference in graphical models, including belief propagation (Smith and Eisner, 2008),
 6089 integer linear programming (Martins et al., 2009), variational inference (Martins et al.,
 6090 2010), and Markov Chain Monte Carlo (Zhang et al., 2014).

6091 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:

$$\text{Linear} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \mathbf{f}(i \xrightarrow{r} j, \mathbf{w}) \quad [11.4]$$

$$\text{Neural} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \text{Feedforward}([\mathbf{u}_{w_i}; \mathbf{u}_{w_j}]; \boldsymbol{\theta}) \quad [11.5]$$

$$\text{Generative} \quad \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) = \log p(w_j, r | w_i). \quad [11.6]$$

6092 Linear feature-based arc scores

6093 Linear models for dependency parsing incorporate many of the same features used in
 6094 sequence labeling and discriminative constituent parsing. These include:

- 6095 • the length and direction of the arc;
- 6096 • the words w_i and w_j linked by the dependency relation;
- 6097 • the prefixes, suffixes, and parts-of-speech of these words;
- 6098 • the neighbors of the dependency arc, $w_{i-1}, w_{i+1}, w_{j-1}, w_{j+1}$;
- 6099 • the prefixes, suffixes, and part-of-speech of these neighbor words.

6100 Each of these features can be conjoined with the dependency edge label r . Note that
 6101 features in an arc-factored parser can refer to words other than w_i and w_j . The restriction
 6102 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:

$$\begin{aligned} f(3 \rightarrow 5, \text{we eat sushi with chopsticks}) = & \langle \text{sushi} \rightarrow \text{chopsticks}, \\ & \text{sushi} \rightarrow \text{NNS}, \\ & \text{NN} \rightarrow \text{chopsticks}, \\ & \text{NNS} \rightarrow \text{NN} \rangle. \end{aligned}$$

6103 Regularized discriminative learning algorithms can then trade off between features at
 6104 varying levels of detail. McDonald et al. (2005) take this approach as far as *tetralexical*
 6105 features (e.g., $(w_i, w_{i+1}, w_{j-1}, w_j)$). Such features help to avoid choosing arcs that are un-
 6106 likely due to the intervening words: for example, there is unlikely to be an edge between
 6107 two nouns if the intervening span contains a verb. A large list of first and second-order
 6108 features is provided by Bohnet (2010), who uses a hashing function to store these features
 6109 efficiently.

6110 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,

$$\mathbf{z} = g(\boldsymbol{\Theta}_r[\mathbf{x}_i; \mathbf{x}_j] + \mathbf{b}_r^{(z)}) \quad [11.8]$$

$$\psi(i \xrightarrow{r} j) = \boldsymbol{\beta}_r \mathbf{z} + \mathbf{b}_r^{(y)}, \quad [11.9]$$

6111 where $\boldsymbol{\Theta}_r$ is a matrix, $\boldsymbol{\beta}_r$ is a vector, each b_r is a scalar, and the function g is an elementwise
 6112 tanh activation function.

6113 The vector \mathbf{x}_i can be set equal to the word embedding, which may be pre-trained or
 6114 learned by backpropagation (Pei et al., 2015). Alternatively, contextual information can
 6115 be incorporated by applying a bidirectional recurrent neural network across the input, as

described in § 7.6. The RNN hidden states at each word can be used as inputs to the arc scoring function (Kiperwasser and Goldberg, 2016).

Feature-based arc scores are computationally expensive, due to the costs of storing and searching a huge table of weights. Neural arc scores can be viewed as a compact solution to this problem. Rather than working in the space of tuples of lexical features, the hidden layers of a feedforward network can be viewed as implicitly computing feature combinations, with each layer of the network evaluating progressively more words. An early paper on neural dependency parsing showed substantial speed improvements at test time, while also providing higher accuracy than feature-based models (Chen and Manning, 2014).

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*,

$$\mathbf{y} = \{(ROOT, 2), (2, 1), (2, 3), (3, 5), (5, 4)\} \quad [11.10]$$

$$\log p(\mathbf{w} \mid \mathbf{y}) = \sum_{(i \rightarrow j) \in \mathbf{y}} \log p(w_j \mid w_i) \quad [11.11]$$

$$\begin{aligned} &= \log p(eat \mid ROOT) + \log p(we \mid eat) + \log p(sushi \mid eat) \\ &\quad + \log p(rice \mid sushi) + \log p(with \mid rice). \end{aligned} \quad [11.12]$$

Probabilistic generative models are used in combination with expectation-maximization (chapter 5) for unsupervised dependency parsing (Klein and Manning, 2004).

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,

$$\hat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{y}') \quad [11.13]$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \mathbf{f}(\mathbf{w}, \mathbf{y}) - \mathbf{f}(\mathbf{w}, \hat{\mathbf{y}}) \quad [11.14]$$

In this case, the argmax requires a maximization over all dependency trees for the sentence, which can be computed using the algorithms described in § 11.2.1. We can apply all the usual tricks from § 2.3: weight averaging, a large margin objective, and regularization. McDonald et al. (2005) were the first to treat dependency parsing as a structure

6134 prediction problem, using MIRA, an online margin-based learning algorithm. Neural arc
 6135 scores can be learned in the same way, backpropagating from a margin loss to updates on
 6136 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,

$$p(\mathbf{y} \mid \mathbf{w}) = \frac{\exp \sum_{i \xrightarrow{r} j \in \mathbf{y}} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_{i \xrightarrow{r} j \in \mathbf{y}'} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})} \quad [11.15]$$

6137 Such a model is trained to minimize the negative log conditional-likelihood. Just as in
 6138 CRF sequence models (§ 7.5.3) and the logistic regression classifier (§ 2.5), the gradients
 6139 involve marginal probabilities $p(i \xrightarrow{r} j \mid \mathbf{w}; \boldsymbol{\theta})$, which in this case are probabilities over
 6140 individual dependencies. In arc-factored models, these probabilities can be computed
 6141 in polynomial time. For projective dependency trees, the marginal probabilities can be
 6142 computed in cubic time, using a variant of the inside-outside algorithm (Lari and Young,
 6143 1990). For non-projective dependency parsing, marginals can also be computed in cubic
 6144 time, using the **matrix-tree theorem** (Koo et al., 2007; McDonald et al., 2007; Smith and
 6145 Smith, 2007). Details of these methods are described by Kübler et al. (2009).

6146 11.3 Transition-based dependency parsing

6147 Graph-based dependency parsing offers exact inference, meaning that it is possible to re-
 6148 cover the best-scoring parse for any given model. But this comes at a price: the scoring
 6149 function is required to decompose into local parts — in the case of non-projective parsing,
 6150 these parts are restricted to individual arcs. These limitations are felt more keenly in de-
 6151 pendency parsing than in sequence labeling, because second-order dependency features
 6152 are critical to correctly identify some types of attachments. For example, prepositional
 6153 phrase attachment depends on the attachment point, the object of the preposition, and
 6154 the preposition itself; arc-factored scores cannot account for all three of these features si-
 6155 multaneously. Graph-based dependency parsing may also be criticized on the basis of
 6156 intuitions about human language processing: people read and listen to sentences *sequen-*
 6157 *tially*, incrementally building mental models of the sentence structure and meaning before
 6158 getting to the end (Jurafsky, 1996). This seems hard to reconcile with graph-based algo-
 6159 rithms, which perform bottom-up operations on the entire sentence, requiring the parser
 6160 to keep every word in memory. Finally, from a practical perspective, graph-based depen-
 6161 dency parsing is relatively slow, running in cubic time in the length of the input.

6162 Transition-based algorithms address all three of these objections. They work by mov-
 6163 ing through the sentence sequentially, while performing actions that incrementally up-
 6164 date a stored representation of what has been read thus far. As with the shift-reduce

6165 parser from § 10.6.2, this representation consists of a stack, onto which parsing substructures
 6166 can be pushed and popped. In shift-reduce, these substructures were constituents;
 6167 in the transition systems that follow, they will be projective dependency trees over partial
 6168 spans of the input.⁴ Parsing is complete when the input is consumed and there is only
 6169 a single structure on the stack. The sequence of actions that led to the parse is known as
 6170 the **derivation**. One problem with transition-based systems is that there may be multiple
 6171 derivations for a single parse structure — a phenomenon known as **spurious ambiguity**.

6172 11.3.1 Transition systems for dependency parsing

6173 A **transition system** consists of a representation for describing configurations of the parser,
 6174 and a set of transition actions, which manipulate the configuration. There are two main
 6175 transition systems for dependency parsing: **arc-standard**, which is closely related to shift-
 6176 reduce, and **arc-eager**, which adds an additional action that can simplify derivations (Ab-
 6177 ney and Johnson, 1991). In both cases, transitions are between **configurations** that are
 6178 represented as triples, $C = (\sigma, \beta, A)$, where σ is the stack, β is the input buffer, and A is
 6179 the list of arcs that have been created (Nivre, 2008). In the initial configuration,

$$C_{\text{initial}} = ([\text{ROOT}], \mathbf{w}, \emptyset), \quad [11.16]$$

6180 indicating that the stack contains only the special node ROOT, the entire input is on the
 6181 buffer, and the set of arcs is empty. An accepting configuration is,

$$C_{\text{accept}} = ([\text{ROOT}], \emptyset, A), \quad [11.17]$$

6182 where the stack contains only ROOT, the buffer is empty, and the arcs A define a spanning
 6183 tree over the input. The arc-standard and arc-eager systems define a set of transitions
 6184 between configurations, which are capable of transforming an initial configuration into
 6185 an accepting configuration. In both of these systems, the number of actions required to
 6186 parse an input grows linearly in the length of the input, making transition-based parsing
 6187 considerably more efficient than graph-based methods.

6188 Arc-standard

6189 The **arc-standard** transition system is closely related to shift-reduce, and to the LR algo-
 6190 rithm that is used to parse programming languages (Aho et al., 2006). It includes the
 6191 following classes of actions:

- 6192 • SHIFT: move the first item from the input buffer on to the top of the stack,

$$(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A), \quad [11.18]$$

⁴Transition systems also exist for non-projective dependency parsing (e.g., Nivre, 2008).

6193 where we write $i|\beta$ to indicate that i is the leftmost item in the input buffer, and $\sigma|i$
 6194 to indicate the result of pushing i on to stack σ .

- 6195 • ARC-LEFT: create a new left-facing arc of type r between the item on the top of the
 6196 stack and the first item in the input buffer. The head of this arc is j , which remains
 6197 at the front of the input buffer. The arc $j \xrightarrow{r} i$ is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \oplus j \xrightarrow{r} i), \quad [11.19]$$

6198 where r is the label of the dependency arc, and \oplus concatenates the new arc $j \xrightarrow{r} i$ to
 6199 the list A .

- 6200 • ARC-RIGHT: creates a new right-facing arc of type r between the item on the top of the
 6201 stack and the first item in the input buffer. The head of this arc is i , which is
 6202 “popped” from the stack and pushed to the front of the input buffer. The arc $i \xrightarrow{r} j$
 6203 is added to A . Formally,

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, i|\beta, A \oplus i \xrightarrow{r} j), \quad [11.20]$$

6204 where again r is the label of the dependency arc.

6205 Each action has preconditions. The SHIFT action can be performed only when the buffer
 6206 has at least one element. The ARC-LEFT action cannot be performed when the root node
 6207 ROOT is on top of the stack, since this node must be the root of the entire tree. The ARC-
 6208 LEFT and ARC-RIGHT remove the modifier words from the stack (in the case of ARC-LEFT)
 6209 and from the buffer (in the case of ARC-RIGHT), so it is impossible for any word to have
 6210 more than one parent. Furthermore, the end state can only be reached when every word is
 6211 removed from the buffer and stack, so the set of arcs is guaranteed to constitute a spanning
 6212 tree. An example arc-standard derivation is shown in Table 11.2.

6213 Arc-eager dependency parsing

6214 In the arc-standard transition system, a word is completely removed from the parse once
 6215 it has been made the modifier in a dependency arc. At this time, any dependents of
 6216 this word must have already been identified. Right-branching structures are common in
 6217 English (and many other languages), with words often modified by units such as prepo-
 6218 sitional phrases to their right. In the arc-standard system, this means that we must first
 6219 shift all the units of the input onto the stack, and then work backwards, creating a series of
 6220 arcs, as occurs in Table 11.2. Note that the decision to shift *bagels* onto the stack guarantees
 6221 that the prepositional phrase *with lox* will attach to the noun phrase, and that this decision
 6222 must be made before the prepositional phrase is itself parsed. This has been argued to be
 6223 cognitively implausible (Abney and Johnson, 1991); from a computational perspective, it
 6224 means that a parser may need to look several steps ahead to make the correct decision.

| σ | β | action | arc added to \mathcal{A} |
|---|----------------------------------|-----------|---|
| 1. [ROOT] | <i>they like bagels with lox</i> | SHIFT | |
| 2. [ROOT, <i>they</i>] | <i>like bagels with lox</i> | ARC-LEFT | (<i>they</i> \leftarrow <i>like</i>) |
| 3. [ROOT] | <i>like bagels with lox</i> | SHIFT | |
| 4. [ROOT, <i>like</i>] | <i>bagels with lox</i> | SHIFT | |
| 5. [ROOT, <i>like</i> , <i>bagels</i>] | <i>with lox</i> | SHIFT | |
| 6. [ROOT, <i>like</i> , <i>bagels</i> , <i>with</i>] | <i>lox</i> | ARC-LEFT | (<i>with</i> \leftarrow <i>lox</i>) |
| 7. [ROOT, <i>like</i> , <i>bagels</i>] | <i>lox</i> | ARC-RIGHT | (<i>bagels</i> \rightarrow <i>lox</i>) |
| 8. [ROOT, <i>like</i>] | <i>bagels</i> | ARC-RIGHT | (<i>like</i> \rightarrow <i>bagels</i>) |
| 9. [ROOT] | <i>like</i> | ARC-RIGHT | (ROOT \rightarrow <i>like</i>) |
| 10. [ROOT] | \emptyset | DONE | |

Table 11.2: Arc-standard derivation of the unlabeled dependency parse for the input *they like bagels with lox*.

6225 **Arc-eager dependency parsing** changes the ARC-RIGHT action so that right depen-
 6226 dents can be attached before all of their dependents have been found. Rather than re-
 6227 moving the modifier from both the buffer and stack, the ARC-RIGHT action pushes the
 6228 modifier on to the stack, on top of the head. Because the stack can now contain elements
 6229 that already have parents in the partial dependency graph, two additional changes are
 6230 necessary:

- 6231 • A precondition is required to ensure that the ARC-LEFT action cannot be applied
 6232 when the top element on the stack already has a parent in A .
 6233 • A new REDUCE action is introduced, which can remove elements from the stack if
 6234 they already have a parent in A :

$$(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A). \quad [11.21]$$

6235 As a result of these changes, it is now possible to create the arc *like* \rightarrow *bagels* before parsing
 6236 the prepositional phrase *with lox*. Furthermore, this action does not imply a decision about
 6237 whether the prepositional phrase will attach to the noun or verb. Noun attachment is
 6238 chosen in the parse in Table 11.3, but verb attachment could be achieved by applying the
 6239 REDUCE action at step 5 or 7.

6240 Projectivity

6241 The arc-standard and arc-eager transition systems are guaranteed to produce projective
 6242 dependency trees, because all arcs are between the word at the top of the stack and the

| σ | β | action | arc added to \mathcal{A} |
|---|----------------------------------|-----------|---|
| 1. [ROOT] | <i>they like bagels with lox</i> | SHIFT | |
| 2. [ROOT, <i>they</i>] | <i>like bagels with lox</i> | ARC-LEFT | (<i>they</i> \leftarrow <i>like</i>) |
| 3. [ROOT] | <i>like bagels with lox</i> | 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*.

6243 left-most edge of the buffer (Nivre, 2008). Non-projective transition systems can be con-
 6244 structed by adding actions that create arcs with words that are second or third in the
 6245 stack (Attardi, 2006), or by adopting an alternative configuration structure, which main-
 6246 tains a list of all words that do not yet have heads (Covington, 2001). In **pseudo-projective**
 6247 **dependency parsing**, a projective dependency parse is generated first, and then a set of
 6248 graph transformation techniques are applied, producing non-projective edges (Nivre and
 6249 Nilsson, 2005).

6250 Beam search

6251 In “greedy” transition-based parsing, the parser tries to make the best decision at each
 6252 configuration. This can lead to search errors, when an early decision locks the parser into
 6253 a poor derivation. For example, in Table 11.2, if ARC-RIGHT were chosen at step 4, then
 6254 the parser would later be forced to attach the prepositional phrase *with lox* to the verb
 6255 *likes*. Note that the *likes* \rightarrow *bagels* arc is indeed part of the correct dependency parse, but
 6256 the arc-standard transition system requires it to be created later in the derivation.

Beam search is a general technique for ameliorating search errors in incremental de-
 coding.⁵ While searching, the algorithm maintains a set of partially-complete hypotheses,
 called a beam. At step t of the derivation, there is a set of k hypotheses, each of which

⁵Beam search is used throughout natural language processing, and beyond. In this text, it appears again in coreference resolution (§ 15.2.4) and machine translation (§ 18.4).

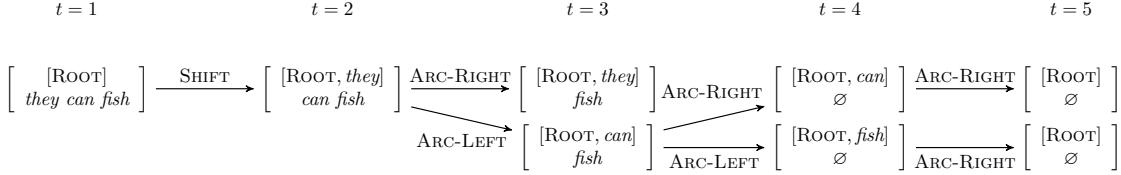


Figure 11.7: Beam search for unlabeled dependency parsing, with beam size $K = 2$. The arc lists for each configuration are not shown, but can be computed from the transitions.

includes a score $s_t^{(k)}$ and a set of dependency arcs $A_t^{(k)}$:

$$h_t^{(k)} = (s_t^{(k)}, A_t^{(k)}) \quad [11.22]$$

6257 Each hypothesis is then “expanded” by considering the set of all valid actions from the
 6258 current configuration $c_t^{(k)}$, written $\mathcal{A}(c_t^{(k)})$. This yields a large set of new hypotheses. For
 6259 each action $a \in \mathcal{A}(c_t^{(k)})$, we score the new hypothesis $A_t^{(k)} \oplus a$. The top k hypotheses
 6260 by this scoring metric are kept, and parsing proceeds to the next step (Zhang and Clark,
 6261 2008). Note that beam search requires a scoring function for action *sequences*, rather than
 6262 individual actions. This issue will be revisited in the next section.

6263 Figure 11.7 shows the application of beam search to dependency parsing, with a beam
 6264 size of $K = 2$. For the first transition, the only valid action is SHIFT, so there is only
 6265 one possible configuration at $t = 2$. From this configuration, there are three possible
 6266 actions. The two best scoring actions are ARC-RIGHT and ARC-LEFT, and so the resulting
 6267 hypotheses from these actions are on the beam at $t = 3$. From these configurations, there
 6268 are three possible actions each, but the best two are expansions of the bottom hypothesis
 6269 at $t = 3$. Parsing continues until $t = 5$, at which point both hypotheses reach an accepting
 6270 state. The best-scoring hypothesis is then selected as the parse.

6271 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,

$$\hat{a} = \underset{a \in \mathcal{A}(c)}{\operatorname{argmax}} \Psi(a, c, \mathbf{w}; \boldsymbol{\theta}), \quad [11.23]$$

6272 where $\mathcal{A}(c)$ is the set of admissible actions in the current configuration c , \mathbf{w} is the input,
 6273 and Ψ is a scoring function with parameters $\boldsymbol{\theta}$ (Yamada and Matsumoto, 2003).

6274 A feature-based score can be computed, $\Psi(a, c, \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(a, c, \mathbf{w})$, using features that
 6275 may consider any aspect of the current configuration and input sequence. Typical features
 6276 for transition-based dependency parsing include: the word and part-of-speech of the top

6277 element on the stack; the word and part-of-speech of the first, second, and third elements
 6278 on the input buffer; pairs and triples of words and parts-of-speech from the top of the
 6279 stack and the front of the buffer; the distance (in tokens) between the element on the top
 6280 of the stack and the element in the front of the input buffer; the number of modifiers of
 6281 each of these elements; and higher-order dependency features as described above in the
 6282 section on graph-based dependency parsing (see, e.g., Zhang and Nivre, 2011).

6283 Parse actions can also be scored by neural networks. For example, Chen and Manning
 6284 (2014) build a feedforward network in which the input layer consists of the concatenation
 6285 of embeddings of several words and tags:

- 6286 • the top three words on the stack, and the first three words on the buffer;
- 6287 • the first and second leftmost and rightmost children (dependents) of the top two
 words on the stack;
- 6288 • the leftmost and right most grandchildren of the top two words on the stack;
- 6289 • 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,

$$z = g(\Theta^{(x \rightarrow z)} \mathbf{x}(c, \mathbf{w})) \quad [11.24]$$

$$\psi(a, c, \mathbf{w}; \theta) = \Theta_a^{(z \rightarrow y)} z, \quad [11.25]$$

6291 where the vector z plays the same role as the features $f(a, c, \mathbf{w})$, but is a learned represen-
 6292 tation. Chen and Manning (2014) use a cubic elementwise activation function, $g(x) = x^3$,
 6293 so that the hidden layer models products across all triples of input features. The learning
 6294 algorithm updates the embeddings as well as the parameters of the feedforward network.

6295 11.3.3 Learning to parse

6296 Transition-based dependency parsing suffers from a mismatch between the supervision,
 6297 which comes in the form of dependency trees, and the classifier's prediction space, which
 6298 is a set of parsing actions. One solution is to create new training data by converting parse
 6299 trees into action sequences; another is to derive supervision directly from the parser's
 6300 performance.

6301 **Oracle-based training**

6302 A transition system can be viewed as a function from action sequences (derivations) to
 6303 parse trees. The inverse of this function is a mapping from parse trees to derivations,
 6304 which is called an **oracle**. For the arc-standard and arc-eager parsing system, an oracle can
 6305 be computed in linear time in the length of the derivation (Kübler et al., 2009, page 32).
 6306 Both the arc-standard and arc-eager transition systems suffer from spurious ambiguity:
 6307 there exist dependency parses for which multiple derivations are possible, such as $1 \leftarrow 2 \rightarrow 3$. The oracle must choose between these different derivations. For example, the
 6308 algorithm described by Kübler et al. (2009) would first create the left arc ($1 \leftarrow 2$), and then
 6309 create the right arc, $(1 \leftarrow 2) \rightarrow 3$; another oracle might begin by shifting twice, resulting
 6310 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):

$$p(a | c, \mathbf{w}) = \frac{\exp \Psi(a, c, \mathbf{w}; \theta)}{\sum_{a' \in \mathcal{A}(c)} \exp \Psi(a', c, \mathbf{w}; \theta)} \quad [11.26]$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^N \sum_{t=1}^{|A^{(i)}|} \log p(a_t^{(i)} | c_t^{(i)}, \mathbf{w}), \quad [11.27]$$

6312 where $|A^{(i)}|$ is the length of the action sequence $A^{(i)}$.

6313 Recall that beam search requires a scoring function for action sequences. Such a score
 6314 can be obtained by adding the log-likelihoods (or hinge losses) across all actions in the
 6315 sequence (Chen and Manning, 2014).

6316 **Global objectives**

6317 The objective in Equation 11.27 is **locally-normalized**: it is the product of normalized
 6318 probabilities over individual actions. A similar characterization could be made of non-
 6319 probabilistic algorithms in which hinge-loss objectives are summed over individual ac-
 6320 tions. In either case, training on individual actions can be sub-optimal with respect to
 6321 global performance, due to the **label bias problem** (Lafferty et al., 2001; Andor et al.,
 6322 2016).

6323 As a stylized example, suppose that a given configuration appears 100 times in the
 6324 training data, with action a_1 as the oracle action in 51 cases, and a_2 as the oracle action in
 6325 the other 49 cases. However, in cases where a_2 is correct, choosing a_1 results in a cascade
 6326 of subsequent errors, while in cases where a_1 is correct, choosing a_2 results in only a single

6327 error. A classifier that is trained on a local objective function will learn to always choose
 6328 a_1 , but choosing a_2 would minimize the overall number of errors.

6329 This observation motivates a global objective, such as the globally-normalized condi-
 6330 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, $\mathbb{A}(\mathbf{w})$.⁶ In the conditional random field model for sequence labeling (§ 7.5.3), it was possible to compute this sum explicitly, using dynamic programming. In transition-based parsing, this is not possible. However, the sum can be approximated using beam search,

$$\sum_{A' \in \mathbb{A}(\mathbf{w})} \exp \sum_{t=1}^{|A'|} \Psi(a'_t, c'_t, \mathbf{w}) \approx \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}), \quad [11.29]$$

where $A^{(k)}$ is an action sequence on a beam of size K . This gives rise to the following loss function,

$$L(\boldsymbol{\theta}) = - \sum_{t=1}^{|A^{(i)}|} \Psi(a_t^{(i)}, c_t^{(i)}, \mathbf{w}) + \log \sum_{k=1}^K \exp \sum_{t=1}^{|A^{(k)}|} \Psi(a_t^{(k)}, c_t^{(k)}, \mathbf{w}). \quad [11.30]$$

6331 The derivatives of this loss involve expectations with respect to a probability distribution
 6332 over action sequences on the beam.

6333 *Early update and the incremental perceptron

6334 When learning in the context of beam search, the goal is to learn a decision function so that
 6335 the gold dependency parse is always reachable from at least one of the partial derivations
 6336 on the beam. (The combination of a transition system (such as beam search) and a scoring
 6337 function for actions is known as a **policy**.) To achieve this, we can make an **early update**
 6338 as soon as the oracle action sequence “falls off” the beam, even before a complete analysis
 6339 is available (Collins and Roark, 2004; Daumé III and Marcu, 2005). The loss can be based
 6340 on the best-scoring hypothesis on the beam, or the sum of all hypotheses (Huang et al.,
 6341 2012).

6342 For example, consider the beam search in Figure 11.7. In the correct parse, *fish* is the
 6343 head of dependency arcs to both of the other two words. In the arc-standard system,

⁶Andor 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.

6344 this can be achieved only by using SHIFT for the first two actions. At $t = 3$, the oracle
 6345 action sequence has fallen off the beam. The parser should therefore stop, and update the
 6346 parameters by the gradient $\frac{\partial}{\partial \theta} L(A_{1:3}^{(i)}, \{A_{1:3}^{(k)}\}; \theta)$, where $A_{1:3}^{(i)}$ is the first three actions of the
 6347 oracle sequence, and $\{A_{1:3}^{(k)}\}$ is the beam.

6348 This integration of incremental search and learning was first developed in the **incre-
 6349 mental perceptron** (Collins and Roark, 2004). This method updates the parameters with
 6350 respect to a hinge loss, which compares the top-scoring hypothesis and the gold action
 6351 sequence, up to the current point t . Several improvements to this basic protocol are pos-
 6352 sible:

- 6353 • As noted earlier, the gold dependency parse can be derived by multiple action se-
 6354 quences. Rather than checking for the presence of a single oracle action sequence on
 6355 the beam, we can check if the gold dependency parse is *reachable* from the current
 6356 beam, using a **dynamic oracle** (Goldberg and Nivre, 2012).
- 6357 • By maximizing the score of the gold action sequence, we are training a decision
 6358 function to find the correct action given the gold context. But in reality, the parser
 6359 will make errors, and the parser is not trained to find the best action given a context
 6360 that may not itself be optimal. This issue is addressed by various generalizations of
 6361 incremental perceptron, known as **learning to search** (Daumé III et al., 2009). Some
 6362 of these methods are discussed in chapter 15.

6363 11.4 Applications

6364 Dependency parsing is used in many real-world applications: any time you want to know
 6365 about pairs of words which might not be adjacent, you can use dependency arcs instead
 6366 of regular expression search patterns. For example, you may want to match strings like
 6367 *delicious pastries*, *delicious French pastries*, and *the pastries are delicious*.

6368 It is possible to search the Google n -grams corpus by dependency edges, finding the
 6369 trend in how often a dependency edge appears over time. For example, we might be inter-
 6370 ested in knowing when people started talking about *writing code*, but we also want *write*
 6371 *some code*, *write good code*, *write all the code*, etc. The result of a search on the dependency
 6372 edge *write* → *code* is shown in Figure 11.8. This capability has been applied to research
 6373 in digital humanities, such as the analysis of gender in Shakespeare Muralidharan and
 6374 Hearst (2013).

6375 A classic application of dependency parsing is **relation extraction**, which is described

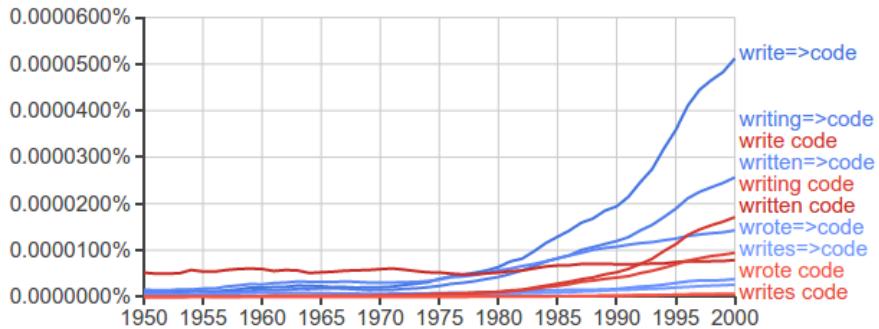


Figure 11.8: Google n-grams results for the bigram *write code* and the dependency arc *write => code* (and their morphological variants)

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),

6375 which stand in some relation to each other (in this case, the relation is authorship). Such
 6376 entity pairs are often referenced via consistent chains of dependency relations. Therefore,
 6377 dependency paths are often a useful feature in supervised systems which learn to detect
 6378 new instances of a relation, based on labeled examples of other instances of the same
 6379 relation type (Culotta and Sorensen, 2004; Fundel et al., 2007; Mintz et al., 2009).

6380 Cui et al. (2005) show how dependency parsing can improve automated question an-
 6381 swering. Suppose you receive the following query:

6382 (11.1) What percentage of the nation's cheese does Wisconsin produce?

6383 The corpus contains this sentence:

6384 (11.2) In Wisconsin, where farmers produce 28% of the nation's cheese, ...

6385 The location of *Wisconsin* in the surface form of this string makes it a poor match for the
 6386 query. However, in the dependency graph, there is an edge from *produce* to *Wisconsin* in
 6387 both the question and the potential answer, raising the likelihood that this span of text is
 6388 relevant to the question.

6389 A final example comes from sentiment analysis. As discussed in chapter 4, the polarity
 6390 of a sentence can be reversed by negation, e.g.

6391 (11.3) *There is no reason at all to believe the polluters will suddenly become reasonable.*

6392 By tracking the sentiment polarity through the dependency parse, we can better iden-
 6393 tify the overall polarity of the sentence, determining when key sentiment words are re-
 6394 versed (Wilson et al., 2005; Nakagawa et al., 2010).

6395 Additional resources

6396 More details on dependency grammar and parsing algorithms can be found in the manuscript
 6397 by Kübler et al. (2009). For a comprehensive but whimsical overview of graph-based de-
 6398 pendency parsing algorithms, see Eisner (1997). Jurafsky and Martin (2018) describe an
 6399 **agenda-based** version of beam search, in which the beam contains hypotheses of varying
 6400 lengths. New hypotheses are added to the beam only if their score is better than the worst
 6401 item currently on the beam. Another search algorithm for transition-based parsing is
 6402 **easy-first**, which abandons the left-to-right traversal order, and adds the highest-scoring
 6403 edges first, regardless of where they appear (Goldberg and Elhadad, 2010). Goldberg et al.
 6404 (2013) note that although transition-based methods can be implemented in linear time in
 6405 the length of the input, naïve implementations of beam search will require quadratic time,
 6406 due to the cost of copying each hypothesis when it is expanded on the beam. This issue
 6407 can be addressed by using a more efficient data structure for the stack.

6408 Exercises

- 6409 1. The dependency structure $1 \leftarrow 2 \rightarrow 3$, with 2 as the root, can be obtained from more
 6410 than one set of actions in arc-standard parsing. List both sets of actions that can
 6411 obtain this parse. Don't forget about the edge $\text{ROOT} \rightarrow 2$.
- 6412 2. This problem develops the relationship between dependency parsing and lexicalized
 6413 context-free parsing. Suppose you have a set of unlabeled arc scores $\{\psi(i \rightarrow j)\}_{i,j=1}^M \cup \{\psi(\text{ROOT} \rightarrow j)\}_{j=1}^M$.
 6414
 - 6415 a) Assuming each word type occurs no more than once in the input ($(i \neq j) \Rightarrow$
 6416 $(w_i \neq w_j)$), how would you construct a weighted lexicalized context-free gram-
 6417 mar so that the score of *any* projective dependency tree is equal to the score of
 6418 some equivalent derivation in the lexicalized context-free grammar?
 - 6419 b) Verify that your method works for the example *They fish*.
 - 6420 c) Does your method require the restriction that each word type occur no more
 6421 than once in the input? If so, why?
 - 6422 d) *If your method required that each word type occur only once in the input,
 6423 show how to generalize it.

- 6424 3. In arc-factored dependency parsing of an input of length M , the score of a parse
 6425 is the sum of M scores, one for each arc. In second order dependency parsing, the
 6426 total score is the sum over many more terms. How many terms are the score of the
 6427 parse for Figure 11.2, using a second-order dependency parser with grandparent
 6428 and sibling features? Assume that a child of ROOT has no grandparent score, and
 6429 that a node with no siblings has no sibling scores.
- 6430 4. a) In the worst case, how many terms can be involved in the score of an input of
 6431 length M , assuming second-order dependency parsing? Describe the structure
 6432 of the worst-case parse. As in the previous problem, assume that there is only
 6433 one child of ROOT, and that it does not have any grandparent scores.
 6434 b) What about third-order dependency parsing?
- 6435 5. Provide the UD-style unlabeled dependency parse for the sentence *Xi-Lan eats shoots*
 6436 *and leaves*, assuming *shoots* is a noun and *leaves* is a verb. Provide arc-standard and
 6437 arc-eager derivations for this dependency parse.
- 6438 6. Compute an upper bound on the number of successful derivations in arc-standard
 6439 shift-reduce parsing for unlabeled dependencies, as a function of the length of the
 6440 input, M . Hint: a lower bound is the number of projective decision trees, $\frac{1}{M+1} \binom{3M-2}{M-1}$ (Zhang,
 6441 2017), where $\binom{a}{b} = \frac{a!}{(a-b)!b!}$.
- 6442 7. The **label bias problem** arises when a decision is locally correct, yet leads to a cas-
 6443 cade of errors in some situations (§ 11.3.3). Design a scenario in which this occurs.
 6444 Specifically:
- 6445 • Assume an arc-standard dependency parser, whose action classifier considers
 6446 only the words at the top of the stack and at the front of the input buffer.
 - 6447 • Design two examples, which both involve a decision with identical features.
 - 6448 – In one example, shift is the correct decision; in the other example, arc-left
 6449 or arc-right is the correct decision.
 - 6450 – In one of the two examples, a mistake should lead to at least two attach-
 6451 ment errors.
 - 6452 – In the other example, a mistake should lead only to a single attachment
 6453 error.

6454 For the following exercises, run a dependency parser, such as Stanford’s CoreNLP
 6455 parser, on a large corpus of text (at least 10^5 tokens), such as `nltk.corpus.webtext`.

- 6456 8. The dependency relation NMOD:POSS indicates possession. Compute the top ten
 6457 words most frequently possessed by each of the following pronouns: *his*, *her*, *our*,
 6458 *my*, *your*, and *their* (inspired by Muralidharan and Hearst, 2013).

9. Count all pairs of words grouped by the CONJ relation. Select all pairs of words (i, j) for which i and j each participate in CONJ relations at least five times. Compute and sort by the **pointwise mutual information**, which is defined in § 14.3 as,

$$\text{PMI}(i, j) = \log \frac{\text{p}(i, j)}{\text{p}(i)\text{p}(j)}. \quad [11.31]$$

Here, $\text{p}(i)$ is the fraction of CONJ relations containing word i (in either position), and $\text{p}(i, j)$ is the fraction of such relations linking i and j (in any order).

10. In § 4.2, we encountered lexical semantic relationships such as **synonymy** (same meaning), **antonymy** (opposite meaning), and **hyponymy** (i is a special case of j). Another relevant relation is **co-hyponymy**, which means that i and j share a hyponym. Of the top 20 pairs identified by PMI in the previous problem, how many participate in synsets that are linked by one of these four relations? Use WORDNET to check for these relations, and count a pair of words if any of their synsets are linked.

6468

Part III

6469

Meaning

6470 Chapter 12

6471 Logical semantics

6472 The previous few chapters have focused on building systems that reconstruct the **syntax**
6473 of natural language — its structural organization — through tagging and parsing. But
6474 some of the most exciting and promising potential applications of language technology
6475 involve going beyond syntax to **semantics** — the underlying meaning of the text:

- 6476 • Answering questions, such as *where is the nearest coffeeshop?* or *what is the middle name*
6477 *of the mother of the 44th President of the United States?*.
- 6478 • Building a robot that can follow natural language instructions to execute tasks.
- 6479 • Translating a sentence from one language into another, while preserving the under-
6480 lying meaning.
- 6481 • Fact-checking an article by searching the web for contradictory evidence.
- 6482 • Logic-checking an argument by identifying contradictions, ambiguity, and unsup-
6483 ported assertions.

6484 Semantic analysis involves converting natural language into a **meaning representa-**
6485 **tion**. To be useful, a meaning representation must meet several criteria:

- 6486 • **c1**: it should be unambiguous: unlike natural language, there should be exactly one
6487 meaning per statement;
- 6488 • **c2**: it should provide a way to link language to external knowledge, observations,
6489 and actions;
- 6490 • **c3**: it should support computational **inference**, so that meanings can be combined
6491 to derive additional knowledge;
- 6492 • **c4**: it should be expressive enough to cover the full range of things that people talk
6493 about in natural language.

6494 Much more than this can be said about the question of how best to represent knowledge
 6495 for computation (e.g., Sowa, 2000), but this chapter will focus on these four criteria.

6496 12.1 Meaning and denotation

6497 The first criterion for a meaning representation is that statements in the representation
 6498 should be unambiguous — they should have only one possible interpretation. Natural
 6499 language does not have this property: as we saw in chapter 10, sentences like *cats scratch*
 6500 *people with claws* have multiple interpretations.

6501 But what does it mean for a statement to be unambiguous? Programming languages
 6502 provide a useful example: the output of a program is completely specified by the rules of
 6503 the language and the properties of the environment in which the program is run. For ex-
 6504 ample, the python code $5 + 3$ will have the output 8, as will the codes $(4 * 4) - (3 * 3) + 1$
 6505 and $((8))$. This output is known as the **denotation** of the program, and can be written
 6506 as,

$$\llbracket 5+3 \rrbracket = \llbracket (4 * 4) - (3 * 3) + 1 \rrbracket = \llbracket ((8)) \rrbracket = 8. \quad [12.1]$$

6507 The denotations of these arithmetic expressions are determined by the meaning of the
 6508 **constants** (e.g., 5, 3) and the **relations** (e.g., $+$, $*$, $(,)$). Now let's consider another snippet
 6509 of python code, `double(4)`. The denotation of this code could be, $\llbracket \text{double}(4) \rrbracket = 8$, or
 6510 it could be $\llbracket \text{double}(4) \rrbracket = 44$ — it depends on the meaning of `double`. This meaning
 6511 is defined in a **world model** \mathcal{M} as an infinite set of pairs. We write the denotation with
 6512 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
 6513 model would also define the (infinite) list of constants, e.g., $\{0, 1, 2, \dots\}$. As long as the
 6514 denotation of string ϕ in model \mathcal{M} can be computed unambiguously, the language can be
 6515 said to be unambiguous.

6516 This approach to meaning is known as **model-theoretic semantics**, and it addresses
 6517 not only criterion *c1* (no ambiguity), but also *c2* (connecting language to external knowl-
 6518 edge, observations, and actions). For example, we can connect a representation of the
 6519 meaning of a statement like *the capital of Georgia* with a world model that includes knowl-
 6520 edge base of geographical facts, obtaining the denotation `Atlanta`. We might populate a
 6521 world model by detecting and analyzing the objects in an image, and then use this world
 6522 model to evaluate **propositions** like *a man is riding a moose*. Another desirable property of
 6523 model-theoretic semantics is that when the facts change, the denotations change too: the
 6524 meaning representation of *President of the USA* would have a different denotation in the
 6525 model \mathcal{M}_{2014} as it would in \mathcal{M}_{2022} .

6526 12.2 Logical representations of meaning

6527 Criterion *c3* requires that the meaning representation support inference — for example,
 6528 automatically deducing new facts from known premises. While many representations
 6529 have been proposed that meet these criteria, the most mature is the language of first-order
 6530 logic.¹

6531 12.2.1 Propositional logic

6532 The bare bones of logical meaning representation are Boolean operations on propositions:

6533 **Propositional symbols.** Greek symbols like ϕ and ψ will be used to represent **proposi-**
 6534 **tions**, which are statements that are either true or false. For example, ϕ may corre-
 6535 spond to the proposition, *bagels are delicious*.

6536 **Boolean operators.** We can build up more complex propositional formulas from Boolean
 6537 operators. These include:

- 6538 • Negation $\neg\phi$, which is true if ϕ is false.
- 6539 • Conjunction, $\phi \wedge \psi$, which is true if both ϕ and ψ are true.
- 6540 • Disjunction, $\phi \vee \psi$, which is true if at least one of ϕ and ψ is true
- 6541 • Implication, $\phi \Rightarrow \psi$, which is true unless ϕ is true and ψ is false. Implication
 6542 has identical truth conditions to $\neg\phi \vee \psi$.
- 6543 • Equivalence, $\phi \Leftrightarrow \psi$, which is true if ϕ and ψ are both true or both false. Equiv-
 6544 alence has identical truth conditions to $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$.

6545 It is not strictly necessary to have all five Boolean operators: readers familiar with
 6546 Boolean logic will know that it is possible to construct all other operators from either the
 6547 NAND (not-and) or NOR (not-or) operators. Nonetheless, it is clearest to use all five
 6548 operators. From the truth conditions for these operators, it is possible to define a number
 6549 of “laws” for these Boolean operators, such as,

- 6550 • *Commutativity*: $\phi \wedge \psi = \psi \wedge \phi$, $\phi \vee \psi = \psi \vee \phi$
- 6551 • *Associativity*: $\phi \wedge (\psi \wedge \chi) = (\phi \wedge \psi) \wedge \chi$, $\phi \vee (\psi \vee \chi) = (\phi \vee \psi) \vee \chi$
- 6552 • *Complementation*: $\phi \wedge \neg\phi = \perp$, $\phi \vee \neg\phi = \top$, where \top indicates a true proposition
 6553 and \perp indicates a false proposition.

¹Alternatives include the “variable-free” representation used in semantic parsing of geographical queries (Zelle and Mooney, 1996) and robotic control (Ge and Mooney, 2005), and dependency-based compositional semantics (Liang et al., 2013).

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,

$$\begin{aligned}\phi \Rightarrow \psi & \quad \text{If the music is loud, Max can't sleep.} \\ \phi & \quad \text{The music is loud.}\end{aligned}$$

we can derive ψ (*Max can't sleep*) by application of **modus ponens**, which is one of a set of **inference rules** that can be derived from more basic laws and used to manipulate propositional formulas. **Automated theorem provers** are capable of applying inference rules to a set of premises to derive desired propositions (Loveland, 2016).

12.2.2 First-order logic

Propositional logic is so named because it treats propositions as its base units. However, the criterion *c4* states that our meaning representation should be sufficiently expressive. Now consider the sentence pair,

(12.1) If anyone is making noise, then Max can't sleep.
Abigail is making noise.

People are capable of making inferences from this sentence pair, but such inferences require formal tools that are beyond propositional logic. To understand the relationship between the statement *anyone is making noise* and the statement *Abigail is making noise*, our meaning representation requires the additional machinery of **first-order logic** (FOL).

In FOL, logical propositions can be constructed from relationships between entities. Specifically, FOL extends propositional logic with the following classes of terms:

Constants. These are elements that name individual entities in the model, such as MAX and ABIGAIL. The denotation of each constant in a model \mathcal{M} is an element in the model, e.g., $[\![\text{MAX}]\!] = m$ and $[\![\text{ABIGAIL}]\!] = a$.

Relations. Relations can be thought of as sets of entities, or sets of tuples. For example, the relation CAN-SLEEP is defined as the set of entities who can sleep, and has the denotation $[\![\text{CAN-SLEEP}]\!] = \{a, m, \dots\}$. To test the truth value of the proposition CAN-SLEEP(MAX), we ask whether $[\![\text{MAX}]\!] \in [\![\text{CAN-SLEEP}]\!]$. Logical relations that are defined over sets of entities are sometimes called *properties*.

Relations may also be ordered tuples of entities. For example BROTHER(MAX,ABIGAIL) expresses the proposition that MAX is the brother of ABIGAIL. The denotation of such relations is a set of tuples, $[\![\text{BROTHER}]\!] = \{(m, a), (x, y), \dots\}$. To test the truth value of the proposition BROTHER(MAX,ABIGAIL), we ask whether the tuple $([\![\text{MAX}]\!], [\![\text{ABIGAIL}]\!])$ is in the denotation $[\![\text{BROTHER}]\!]$.

Using constants and relations, it is possible to express statements like *Max can't sleep* and *Max is Abigail's brother*:

$$\neg\text{CAN-SLEEP}(\text{MAX}) \\ \text{BROTHER}(\text{MAX}, \text{ABIGAIL}).$$

These statements can also be combined using Boolean operators, such as,

$$(\text{BROTHER}(\text{MAX}, \text{ABIGAIL}) \vee \text{BROTHER}(\text{MAX}, \text{STEVE})) \Rightarrow \neg\text{CAN-SLEEP}(\text{MAX}).$$

6583 This fragment of first-order logic permits only statements about specific entities. To
 6584 support inferences about statements like *If anyone is making noise, then Max can't sleep*,
 6585 two more elements must be added to the meaning representation:

6586 **Variables.** Variables are mechanisms for referring to entities that are not locally specified.
 6587 We can then write $\text{CAN-SLEEP}(x)$ or $\text{BROTHER}(x, \text{ABIGAIL})$. In these cases, x is a **free**
 6588 **variable**, meaning that we have not committed to any particular assignment.

6589 **Quantifiers.** Variables are bound by quantifiers. There are two quantifiers in first-order
 6590 logic.²

- 6591 • The **existential quantifier** \exists , which indicates that there must be at least one en-
 6592 tity to which the variable can bind. For example, the statement $\exists x \text{MAKES-NOISE}(x)$
 6593 indicates that there is at least one entity for which MAKES-NOISE is true.
- 6594 • The **universal quantifier** \forall , which indicates that the variable must be able to
 6595 bind to any entity in the model. For example, the statement,

$$\text{MAKES-NOISE}(\text{ABIGAIL}) \Rightarrow (\forall x \neg \text{CAN-SLEEP}(x)) \quad [12.3]$$

6596 asserts that if Abigail makes noise, no one can sleep.

6597 The expressions $\exists x$ and $\forall x$ make x into a **bound variable**. A formula that contains
 6598 no free variables is a **sentence**.

6599 **Functions.** Functions map from entities to entities, e.g., $\llbracket \text{CAPITAL-OF(GEORGIA)} \rrbracket = \llbracket \text{ATLANTA} \rrbracket$.
 6600 With functions, it is convenient to add an equality operator, supporting statements
 6601 like,

$$\forall x \exists y \text{MOTHER-OF}(x) = \text{DAUGHTER-OF}(y). \quad [12.4]$$

²In first-order logic, it is possible to quantify only over entities. In **second-order logic**, it is possible to quantify over properties. This makes it possible to represent statements like *Butch has every property that a good boxer has* (example from Blackburn and Bos, 2005),

$$\forall P \forall x ((\text{GOOD-BOXER}(x) \Rightarrow P(x)) \Rightarrow P(\text{BUTCH})). \quad [12.2]$$

6602 Note that MOTHER-OF is a functional analogue of the relation MOTHER, so that
 6603 $\text{MOTHER-OF}(x) = y$ if $\text{MOTHER}(x, y)$. Any logical formula that uses functions can be
 6604 rewritten using only relations and quantification. For example,

$$\text{MAKES-NOISE}(\text{MOTHER-OF}(\text{ABIGAIL})) \quad [12.5]$$

6605 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:

$$\forall x \exists y \text{ SPEAKS}(x, y) \quad [12.6]$$

$$\exists y \forall x \text{ SPEAKS}(x, y). \quad [12.7]$$

6606 In the first case, y may refer to several different languages, while in the second case, there
 6607 is a single y that is spoken by everyone.

6608 Truth-conditional semantics

6609 One way to look at the meaning of an FOL sentence ϕ is as a set of **truth conditions**,
 6610 or models under which ϕ is satisfied. But how to determine whether a sentence is true
 6611 or false in a given model? We will approach this inductively, starting with a predicate
 6612 applied to a tuple of constants. The truth of such a sentence depends on whether the
 6613 tuple of denotations of the constants is in the denotation of the predicate. For example,
 6614 $\text{CAPITAL}(\text{GEORGIA}, \text{ATLANTA})$ is true in model \mathcal{M} iff,

$$(\llbracket \text{GEORGIA} \rrbracket_{\mathcal{M}}, \llbracket \text{ATLANTA} \rrbracket_{\mathcal{M}}) \in \llbracket \text{CAPITAL} \rrbracket_{\mathcal{M}}. \quad [12.8]$$

6615 The Boolean operators \wedge, \vee, \dots provide ways to construct more complicated sentences,
 6616 and the truth of such statements can be assessed based on the truth tables associated with
 6617 these operators. The statement $\exists x \phi$ is true if there is some assignment of the variable x
 6618 to an entity in the model such that ϕ is true; the statement $\forall x \phi$ is true if ϕ is true under
 6619 all possible assignments of x . More formally, we would say that ϕ is **satisfied** under \mathcal{M} ,
 6620 written as $\mathcal{M} \models \phi$.

6621 Truth conditional semantics allows us to define several other properties of sentences
 6622 and pairs of sentences. Suppose that in every \mathcal{M} under which ϕ is satisfied, another
 6623 formula ψ is also satisfied; then ϕ **entails** ψ , which is also written as $\phi \models \psi$. For example,

$$\text{CAPITAL}(\text{GEORGIA}, \text{ATLANTA}) \models \exists x \text{CAPITAL}(\text{GEORGIA}, x). \quad [12.9]$$

6624 A statement that is satisfied under any model, such as $\phi \vee \neg\phi$, is **valid**, written $\models (\phi \vee$
 6625 $\neg\phi)$. A statement that is not satisfied under any model, such as $\phi \wedge \neg\phi$, is **unsatisfiable**,

6626 or **inconsistent**. A **model checker** is a program that determines whether a sentence ϕ
6627 is satisfied in \mathcal{M} . A **model builder** is a program that constructs a model in which ϕ
6628 is satisfied. The problems of checking for consistency and validity in first-order logic
6629 are **undecidable**, meaning that there is no algorithm that can automatically determine
6630 whether an FOL formula is valid or inconsistent.

6631 **Inference in first-order logic**

6632 Our original goal was to support inferences that combine general statements *If anyone is*
6633 *making noise, then Max can't sleep* with specific statements like *Abigail is making noise*. We
6634 can now represent such statements in first-order logic, but how are we to perform the
6635 inference that *Max can't sleep*? One approach is to use “generalized” versions of proposi-
6636 tional inference rules like modus ponens, which can be applied to FOL formulas. By
6637 repeatedly applying such inference rules to a knowledge base of facts, it is possible to
6638 produce proofs of desired propositions. To find the right sequence of inferences to derive
6639 a desired theorem, classical artificial intelligence search algorithms like backward chain-
6640 ing can be applied. Such algorithms are implemented in interpreters for the `prolog` logic
6641 programming language (Pereira and Shieber, 2002).

6642 **12.3 Semantic parsing and the lambda calculus**

6643 The previous section laid out a lot of formal machinery; the remainder of this chapter
6644 links these formalisms back to natural language. Given an English sentence like *Alex likes*
6645 *Brit*, how can we obtain the desired first-order logical representation, `LIKES(ALEX,BRIT)`?
6646 This is the task of **semantic parsing**. Just as a syntactic parser is a function from a natu-
6647 ral language sentence to a syntactic structure such as a phrase structure tree, a semantic
6648 parser is a function from natural language to logical formulas.

6649 As in syntactic analysis, semantic parsing is difficult because the space of inputs and
6650 outputs is very large, and their interaction is complex. Our best hope is that, like syntactic
6651 parsing, semantic parsing can somehow be decomposed into simpler sub-problems. This
6652 idea, usually attributed to the German philosopher Gottlob Frege, is called the **principle**
6653 **of compositionality**: the meaning of a complex expression is a function of the meanings of
6654 that expression's constituent parts. We will define these “constituent parts” as syntactic
6655 constituents: noun phrases and verb phrases. These constituents are combined using
6656 function application: if the syntactic parse contains the production $x \rightarrow y z$, then the
6657 semantics of x , written $x.\text{sem}$, will be computed as a function of the semantics of the

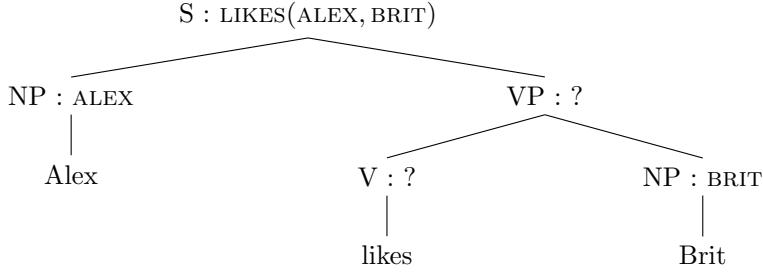


Figure 12.1: The principle of compositionality requires that we identify meanings for the constituents *likes* and *likes Brit* that will make it possible to compute the meaning for the entire sentence.

6658 constituents, $y.\text{sem}$ and $z.\text{sem}$.³ ⁴

6659 12.3.1 The lambda calculus

6660 Let's see how this works for a simple sentence like *Alex likes Brit*, whose syntactic structure
 6661 is shown in Figure 12.1. Our goal is the formula, $\text{LIKES}(\text{ALEX}, \text{BRIT})$, and it is clear that the
 6662 meaning of the constituents *Alex* and *Brit* should be *ALEX* and *BRIT*. That leaves two more
 6663 constituents: the verb *likes*, and the verb phrase *likes Brit*. The meanings of these units
 6664 must be defined in a way that makes it possible to recover the desired meaning for the
 6665 entire sentence by function application. If the meanings of *Alex* and *Brit* are constants,
 6666 then the meanings of *likes* and *likes Brit* must be functional expressions, which can be
 6667 applied to their siblings to produce the desired analyses.

6668 Modeling these partial analyses requires extending the first-order logic meaning rep-
 6669 resentation. We do this by adding **lambda expressions**, which are descriptions of anony-
 6670 mous functions,⁵ e.g.,

$$\lambda x. \text{LIKES}(x, \text{BRIT}). \quad [12.10]$$

6671 This functional expression is the meaning of the verb phrase *likes Brit*; it takes a single
 6672 argument, and returns the result of substituting that argument for x in the expression

³§ 9.3.2 briefly discusses Combinatory Categorial Grammar (CCG) as an alternative to a phrase-structure analysis of syntax. CCG is argued to be particularly well-suited to semantic parsing (Hockenmaier and Steedman, 2007), and is used in much of the contemporary work on machine learning for semantic parsing, summarized in § 12.4.

⁴The approach of algorithmically building up meaning representations from a series of operations on the syntactic structure of a sentence is generally attributed to the philosopher Richard Montague, who published a series of influential papers on the topic in the early 1970s (e.g., Montague, 1973).

⁵Formally, all first-order logic formulas are lambda expressions; in addition, if ϕ is a lambda expression, then $\lambda x.\phi$ is also a lambda expression. Readers who are familiar with functional programming will recognize lambda expressions from their use in programming languages such as Lisp and Python.

6673 $\text{LIKES}(x, \text{BRIT})$. We write this substitution as,

$$(\lambda x.\text{LIKES}(x, \text{BRIT}))@\text{ALEX} = \text{LIKES}(\text{ALEX}, \text{BRIT}), \quad [12.11]$$

6674 with the symbol “@” indicating function application. Function application in the lambda
 6675 calculus is sometimes called **β -reduction** or β -conversion. The expression $\phi@\psi$ indicates
 6676 a function application to be performed by β -reduction, and $\phi(\psi)$ indicates a function or
 6677 predicate in the final logical form.

6678 Equation 12.11 shows how to obtain the desired semantics for the sentence *Alex likes*
 6679 *Brit*: by applying the lambda expression $\lambda x.\text{LIKES}(x, \text{BRIT})$ to the logical constant ALEX.
 6680 This rule of composition can be specified in a **syntactic-semantic grammar**, in which
 6681 syntactic productions are paired with semantic operations. For the syntactic production
 6682 $S \rightarrow \text{NP VP}$, we have the semantic rule $\text{VP.sem}@\text{NP.sem}$.

The meaning of the transitive verb phrase *likes Brit* can also be obtained by function application on its syntactic constituents. For the syntactic production $\text{VP} \rightarrow \text{V NP}$, we apply the semantic rule,

$$\text{VP.sem} = (\text{V.sem})@\text{NP.sem} \quad [12.12]$$

$$= (\lambda y.\lambda x.\text{LIKES}(x, y)) @ (\text{BRIT}) \quad [12.13]$$

$$= \lambda x.\text{LIKES}(x, \text{BRIT}). \quad [12.14]$$

6683 Thus, the meaning of the transitive verb *likes* is a lambda expression whose output is
 6684 *another* lambda expression: it takes y as an argument to fill in one of the slots in the LIKES
 6685 relation, and returns a lambda expression that is ready to take an argument to fill in the
 6686 other slot.⁶

6687 Table 12.1 shows a minimal syntactic-semantic grammar fragment, G_1 . The complete
 6688 **derivation** of *Alex likes Brit* in G_1 is shown in Figure 12.2. In addition to the transitive
 6689 verb *likes*, the grammar also includes the intransitive verb *sleeps*; it should be clear how
 6690 to derive the meaning of sentences like *Alex sleeps*. For verbs that can be either transitive
 6691 or intransitive, such as *eats*, we would have two terminal productions, one for each sense
 6692 (terminal productions are also called the **lexical entries**). Indeed, most of the grammar is
 6693 in the **lexicon** (the terminal productions), since these productions select the basic units of
 6694 the semantic interpretation.

6695 12.3.2 Quantification

6696 Things get more complicated when we move from sentences about named entities to sen-
 6697 tences that involve more general noun phrases. Let’s consider the example, *A dog sleeps*,

⁶This can be written in a few different ways. The notation $\lambda y, x.\text{LIKES}(x, y)$ is a somewhat informal way to indicate a lambda expression that takes two arguments; this would be acceptable in functional programming. Logicians (e.g., Carpenter, 1997) often prefer the more formal notation $\lambda y.\lambda x.\text{LIKES}(x)(y)$, indicating that each lambda expression takes exactly one argument.

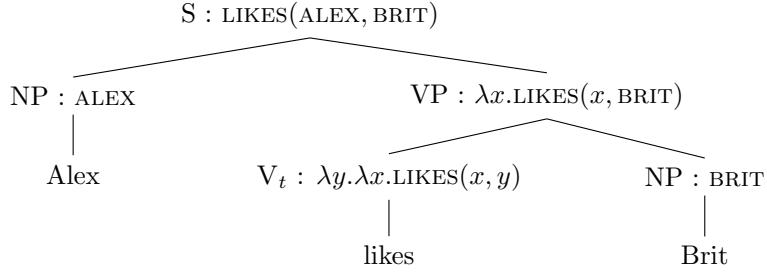


Figure 12.2: Derivation of the semantic representation for *Alex likes Brit* in the grammar G_1 .

| | | | |
|----------------|---------------|-------------------|--|
| S | \rightarrow | NP VP | VP.sem@NP.sem |
| VP | \rightarrow | V _t NP | V _t .sem@NP.sem |
| VP | \rightarrow | V _i | V _i .sem |
| V _t | \rightarrow | likes | $\lambda y. \lambda x. \text{LIKES}(x, y)$ |
| V _i | \rightarrow | sleeps | $\lambda x. \text{SLEEPS}(x)$ |
| NP | \rightarrow | Alex | ALEX |
| NP | \rightarrow | Brit | BRIT |

Table 12.1: G_1 , a minimal syntactic-semantic context-free grammar

which has the meaning $\exists x \text{DOG}(x) \wedge \text{SLEEPS}(x)$. Clearly, the DOG relation will be introduced by the word *dog*, and the SLEEP relation will be introduced by the word *sleeps*. The existential quantifier \exists must be introduced by the lexical entry for the determiner *a*.⁷ However, this seems problematic for the compositional approach taken in the grammar G_1 : if the semantics of the noun phrase *a dog* is an existentially quantified expression, how can it be the argument to the semantics of the verb *sleeps*, which expects an entity? And where does the logical conjunction come from?

There are a few different approaches to handling these issues.⁸ We will begin by reversing the semantic relationship between subject NPs and VPs, so that the production $S \rightarrow \text{NP VP}$ has the semantics $\text{NP.sem}@\text{VP.sem}$: the meaning of the sentence is now the semantics of the noun phrase applied to the verb phrase. The implications of this change are best illustrated by exploring the derivation of the example, shown in Figure 12.3. Let's

⁷Conversely, the sentence *Every dog sleeps* would involve a universal quantifier, $\forall x \text{DOG}(x) \Rightarrow \text{SLEEPS}(x)$. The definite article *the* requires more consideration, since *the dog* must refer to some dog which is uniquely identifiable, perhaps from contextual information external to the sentence. Carpenter (1997, pp. 96-100) summarizes recent approaches to handling definite descriptions.

⁸Carpenter (1997) offers an alternative treatment based on combinatory categorial grammar.

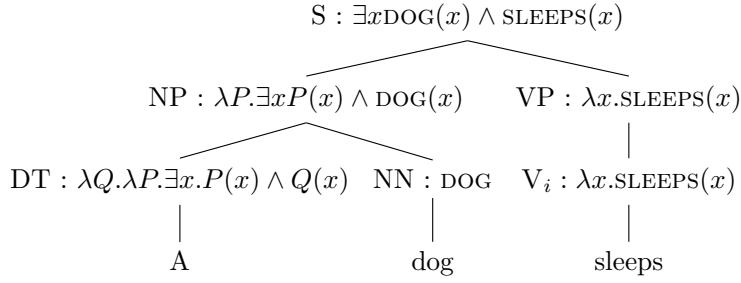


Figure 12.3: Derivation of the semantic representation for *A dog sleeps*, in grammar G_2

6710 start with the indefinite article *a*, to which we assign the rather intimidating semantics,

$$\lambda P.\lambda Q.\exists xP(x) \wedge Q(x). \quad [12.15]$$

This is a lambda expression that takes two **relations** as arguments, P and Q . The relation P is scoped to the outer lambda expression, so it will be provided by the immediately adjacent noun, which in this case is DOG. Thus, the noun phrase *a dog* has the semantics,

$$\text{NP.sem} = \text{DET.sem}@\text{NN.sem} \quad [12.16]$$

$$= (\lambda P.\lambda Q.\exists xP(x) \wedge Q(x)) @ (\text{DOG}) \quad [12.17]$$

$$= \lambda Q.\exists x\text{DOG}(x) \wedge Q(x). \quad [12.18]$$

6711 This is a lambda expression that is expecting another relation, Q , which will be provided
6712 by the verb phrase, SLEEPS. This gives the desired analysis, $\exists x\text{DOG}(x) \wedge \text{SLEEPS}(x)$.⁹

6713 If noun phrases like *a dog* are interpreted as lambda expressions, then proper nouns
6714 like *Alex* must be treated in the same way. This is achieved by **type-raising** from con-
6715 stants to lambda expressions, $x \Rightarrow \lambda P.P(x)$. After type-raising, the semantics of *Alex* is
6716 $\lambda P.P(\text{ALEX})$ — a lambda expression that expects a relation to tell us something about
6717 *ALEX*.¹⁰ Again, make sure you see how the analysis in Figure 12.3 can be applied to the
6718 sentence *Alex sleeps*.

⁹When applying β -reduction to arguments that are themselves lambda expressions, be sure to use unique variable names to avoid confusion. For example, it is important to distinguish the x in the semantics for *a* from the x in the semantics for *likes*. Variable names are abstractions, and can always be changed — this is known as **α -conversion**. For example, $\lambda x.P(x)$ can be converted to $\lambda y.P(y)$, etc.

¹⁰Compositional semantic analysis is often supported by **type systems**, which make it possible to check whether a given function application is valid. The base types are entities e and truth values t . A property, such as DOG, is a function from entities to truth values, so its type is written $\langle e, t \rangle$. A transitive verb has type $\langle e, \langle e, t \rangle \rangle$: after receiving the first entity (the direct object), it returns a function from entities to truth values, which will be applied to the subject of the sentence. The type-raising operation $x \Rightarrow \lambda P.P(x)$ corresponds to a change in type from e to $\langle \langle e, t \rangle, t \rangle$: it expects a function from entities to truth values, and returns a truth value.

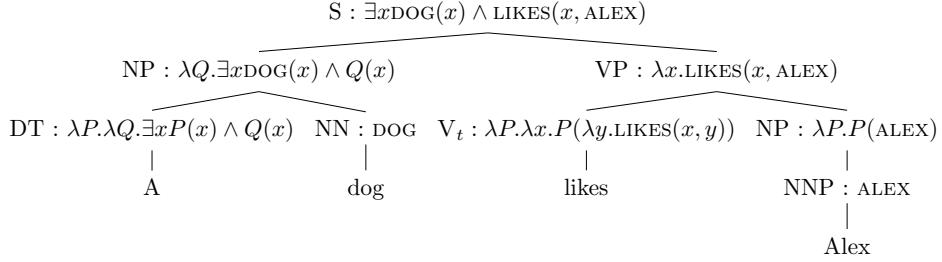


Figure 12.4: Derivation of the semantic representation for *A dog likes Alex*.

6719 Direct objects are handled by applying the same type-raising operation to transitive
 6720 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,

$$\begin{aligned}
 \text{VP.sem} &= \text{V.sem}@\text{NP.sem} \\
 &= (\lambda P. \lambda x. P(\lambda y. \text{LIKES}(x, y))) @ (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) \\
 &= \lambda x. (\lambda Q. \exists z \text{DOG}(z) \wedge Q(z)) @ (\lambda y. \text{LIKES}(x, y)) \\
 &= \lambda x. \exists z \text{DOG}(z) \wedge (\lambda y. \text{LIKES}(x, y)) @ z \\
 &= \lambda x. \exists z \text{DOG}(z) \wedge \text{LIKES}(x, z).
 \end{aligned}$$

6721 These changes are summarized in the revised grammar G_2 , shown in Table 12.2. Figure 6722 12.4 shows a derivation that involves a transitive verb, an indefinite noun phrase, and 6723 a proper noun.

6724 12.4 Learning semantic parsers

6725 As with syntactic parsing, any syntactic-semantic grammar with sufficient coverage risks
 6726 producing many possible analyses for any given sentence. Machine learning is the dom-
 6727 inant approach to selecting a single analysis. We will focus on algorithms that learn to
 6728 score logical forms by attaching weights to features of their derivations (Zettlemoyer
 6729 and Collins, 2005). Alternative approaches include transition-based parsing (Zelle and
 6730 Mooney, 1996; Misra and Artzi, 2016) and methods inspired by machine translation (Wong
 6731 and Mooney, 2006). Methods also differ in the form of supervision used for learning,
 6732 which can range from complete derivations to much more limited training signals. We
 6733 will begin with the case of complete supervision, and then consider how learning is still
 6734 possible even when seemingly key information is missing.

| | | |
|----------------|---------------------------------|--|
| S | \rightarrow NP VP | NP.sem@VP.sem |
| VP | \rightarrow V _t NP | V _t .sem@NP.sem |
| VP | \rightarrow V _i | V _i .sem |
| NP | \rightarrow DET NN | DET.sem@NN.sem |
| NP | \rightarrow NNP | $\lambda P.P(NNP.sem)$ |
| DET | $\rightarrow a$ | $\lambda P.\lambda Q.\exists xP(x) \wedge Q(x)$ |
| DET | $\rightarrow every$ | $\lambda P.\lambda Q.\forall x(P(x) \Rightarrow Q(x))$ |
| V _t | $\rightarrow likes$ | $\lambda P.\lambda x.P(\lambda y.LIKES(x,y))$ |
| V _i | $\rightarrow sleeps$ | $\lambda x.SLEEPS(x)$ |
| NN | $\rightarrow dog$ | DOG |
| NNP | $\rightarrow Alex$ | ALEX |
| NNP | $\rightarrow Brit$ | BRIT |

Table 12.2: G_2 , a syntactic-semantic context-free grammar fragment, which supports quantified noun phrases

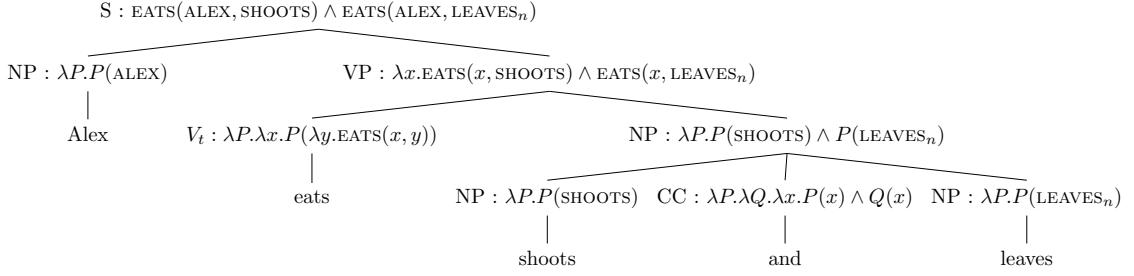
6735 **Datasets** Early work on semantic parsing focused on natural language expressions of
 6736 geographical database queries, such as *What states border Texas*. The GeoQuery dataset
 6737 of Zelle and Mooney (1996) was originally coded in prolog, but has subsequently been
 6738 expanded and converted into the SQL database query language by Popescu et al. (2003)
 6739 and into first-order logic with lambda calculus by Zettlemoyer and Collins (2005), pro-
 6740 viding logical forms like $\lambda x.\text{STATE}(x) \wedge \text{BORDERS}(x, \text{TEXAS})$. Another early dataset con-
 6741 sists of instructions for RoboCup robot soccer teams (Kate et al., 2005). More recent work
 6742 has focused on broader domains, such as the Freebase database (Bollacker et al., 2008),
 6743 for which queries have been annotated by Krishnamurthy and Mitchell (2012) and Cai
 6744 and Yates (2013). Other recent datasets include child-directed speech (Kwiatkowski et al.,
 6745 2012) and elementary school science exams (Krishnamurthy, 2016).

6746 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:

$$\begin{aligned} w^{(i)} &= \text{Alex eats shoots and leaves} \\ y^{(i)} &= \text{EATS(ALEX,SHOOTS)} \wedge \text{EATS(ALEX,LEAVES)} \end{aligned}$$

6747 In the standard supervised learning paradigm that was introduced in § 2.3, we first de-
 6748 fine a feature function, $f(w, y)$, and then learn weights on these features, so that $y^{(i)} =$
 6749 $\operatorname{argmax}_y \theta \cdot f(w, y)$. The weight vector θ is learned by comparing the features of the true
 6750 label $f(w^{(i)}, y^{(i)})$ against either the features of the predicted label $f(w^{(i)}, \hat{y})$ (perceptron,

Figure 12.5: Derivation for gold semantic analysis of *Alex eats shoots and leaves*

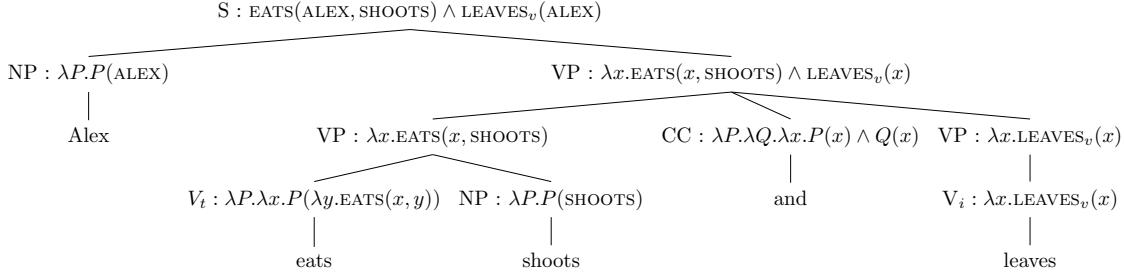
support vector machine) or the expected feature vector $E_{\mathbf{y}|\mathbf{w}}[\mathbf{f}(\mathbf{w}^{(i)}, \mathbf{y})]$ (logistic regression).

While this basic framework seems similar to discriminative syntactic parsing, there is a crucial difference. In (context-free) syntactic parsing, the annotation $\mathbf{y}^{(i)}$ contains all of the syntactic productions; indeed, the task of identifying the correct set of productions is identical to the task of identifying the syntactic structure. In semantic parsing, this is not the case: the logical form $EATS(ALEX, SHOOTS) \wedge EATS(ALEX, LEAVES)$ does not reveal the syntactic-semantic productions that were used to obtain it. Indeed, there may be **spurious ambiguity**, so that a single logical form can be reached by multiple derivations. (We previously encountered spurious ambiguity in transition-based dependency parsing, § 11.3.2.)

These ideas can be formalized by introducing an additional variable \mathbf{z} , representing the derivation of the logical form \mathbf{y} from the text \mathbf{w} . Assume that the feature function decomposes across the productions in the derivation, $\mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) = \sum_{t=1}^T \mathbf{f}(\mathbf{w}, z_t, \mathbf{y})$, where z_t indicates a single syntactic-semantic production. For example, we might have a feature for the production $S \rightarrow NP VP : NP.sem@VP.sem$, as well as for terminal productions like $NNP \rightarrow Alex : ALEX$. Under this decomposition, it is possible to compute scores for each semantically-annotated subtree in the analysis of \mathbf{w} , so that bottom-up parsing algorithms like CKY (§ 10.1) can be applied to find the best-scoring semantic analysis.

Figure 12.5 shows a derivation of the correct semantic analysis of the sentence *Alex eats shoots and leaves*, in a simplified grammar in which the plural noun phrases *shoots* and *leaves* are interpreted as logical constants *SHOOTS* and *LEAVES_n*. Figure 12.6 shows a derivation of an incorrect analysis. Assuming one feature per production, the perceptron update is shown in Table 12.3. From this update, the parser would learn to prefer the noun interpretation of *leaves* over the verb interpretation. It would also learn to prefer noun phrase coordination over verb phrase coordination.

While the update is explained in terms of the perceptron, it would be easy to replace the perceptron with a conditional random field. In this case, the online updates would be

Figure 12.6: Derivation for incorrect semantic analysis of *Alex eats shoots and leaves*

| | | |
|--|---|----|
| $NP_1 \rightarrow NP_2 \text{ CC } NP_3$ | $(CC.\text{sem} @ (NP_2.\text{sem})) @ (NP_3.\text{sem})$ | +1 |
| $VP_1 \rightarrow VP_2 \text{ CC } VP_3$ | $(CC.\text{sem} @ (VP_2.\text{sem})) @ (VP_3.\text{sem})$ | -1 |
| $NP \rightarrow leaves$ | LEAVES_n | +1 |
| $VP \rightarrow V_i$ | $V_i.\text{sem}$ | -1 |
| $V_i \rightarrow leaves$ | $\lambda x.\text{LEAVES}_v$ | -1 |

Table 12.3: Perceptron update for analysis in Figure 12.5 (gold) and Figure 12.6 (predicted)

6779 based on feature expectations, which can be computed using the inside-outside algorithm
 6780 (§ 10.6).

6781 12.4.2 Learning from logical forms

Complete derivations are expensive to annotate, and are rarely available.¹¹ One solution is to focus on learning from logical forms directly, while treating the derivations as **latent variables** (Zettlemoyer and Collins, 2005). In a conditional probabilistic model over logical forms y and derivations z , we have,

$$p(y, z | w) = \frac{\exp(\theta \cdot f(w, z, y))}{\sum_{y', z'} \exp(\theta \cdot f(w, z', y'))}, \quad [12.20]$$

6782 which is the standard log-linear model, applied to the logical form y and the derivation
 6783 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,

$$p(y | w) = \sum_z p(y, z | w). \quad [12.21]$$

¹¹An exception is the work of Ge and Mooney (2005), who annotate the meaning of each syntactic constituents for several hundred sentences.

The semantic parser can then select the logical form with the maximum log marginal probability,

$$\log \sum_z p(\mathbf{y}, \mathbf{z} \mid \mathbf{w}) = \log \sum_z \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}))}{\sum_{\mathbf{y}', \mathbf{z}' \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}'))}} \quad [12.22]$$

$$\propto \log \sum_z \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}')) \quad [12.23]$$

$$\geq \max_z \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}). \quad [12.24]$$

6784 It is impossible to push the log term inside the sum over \mathbf{z} , so our usual linear scoring
 6785 function does not apply. We can recover this scoring function only in approximation, by
 6786 taking the max (rather than the sum) over derivations \mathbf{z} , which provides a lower bound.

Learning can be performed by maximizing the log marginal likelihood,

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^N \log p(\mathbf{y}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}) \quad [12.25]$$

$$= \sum_{i=1}^N \log \sum_{\mathbf{z}} p(\mathbf{y}^{(i)}, \mathbf{z}^{(i)} \mid \mathbf{w}^{(i)}; \boldsymbol{\theta}). \quad [12.26]$$

6787 This log-likelihood is not **convex** in $\boldsymbol{\theta}$, unlike the log-likelihood of a fully-observed condi-
 6788 tional random field. This means that learning can give different results depending on the
 6789 initialization.

The derivative of Equation 12.26 is,

$$\frac{\partial \ell_i}{\partial \boldsymbol{\theta}} = \sum_{\mathbf{z}} p(\mathbf{z} \mid \mathbf{y}, \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - \sum_{\mathbf{y}', \mathbf{z}'} p(\mathbf{y}', \mathbf{z}' \mid \mathbf{w}; \boldsymbol{\theta}) \mathbf{f}(\mathbf{w}, \mathbf{z}', \mathbf{y}') \quad [12.27]$$

$$= E_{\mathbf{z} \mid \mathbf{y}, \mathbf{w}} \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) - E_{\mathbf{y}, \mathbf{z} \mid \mathbf{w}} \mathbf{f}(\mathbf{w}, \mathbf{z}, \mathbf{y}) \quad [12.28]$$

6790 Both expectations can be computed via bottom-up algorithms like inside-outside. Al-
 6791 ternatively, we can again maximize rather than marginalize over derivations for an ap-
 6792 proximate solution. In either case, the first term of the gradient requires us to identify
 6793 derivations \mathbf{z} that are compatible with the logical form \mathbf{y} . This can be done in a bottom-
 6794 up dynamic programming algorithm, by having each cell in the table $t[i, j, X]$ include the
 6795 set of all possible logical forms for $X \sim \mathbf{w}_{i+1:j}$. The resulting table may therefore be much
 6796 larger than in syntactic parsing. This can be controlled by using pruning to eliminate in-
 6797 termediate analyses that are incompatible with the final logical form \mathbf{y} (Zettlemoyer and
 6798 Collins, 2005), or by using beam search and restricting the size of each cell to some fixed
 6799 constant (Liang et al., 2013).

6800 If we replace each expectation in Equation 12.28 with argmax and then apply stochastic
 6801 gradient descent to learn the weights, we obtain the **latent variable perceptron**, a simple

Algorithm 16 Latent variable perceptron

```

1: procedure LATENTVARIABLEPERCEPTRON( $w^{(1:N)}, y^{(1:N)}$ )
2:    $\theta \leftarrow 0$ 
3:   repeat
4:     Select an instance  $i$ 
5:      $z^{(i)} \leftarrow \text{argmax}_z \theta \cdot f(w^{(i)}, z, y^{(i)})$ 
6:      $\hat{y}, \hat{z} \leftarrow \text{argmax}_{y', z'} \theta \cdot f(w^{(i)}, z', y')$ 
7:      $\theta \leftarrow \theta + f(w^{(i)}, z^{(i)}, y^{(i)}) - f(w^{(i)}, \hat{z}, \hat{y})$ 
8:   until tired
9:   return  $\theta$ 

```

and general algorithm for learning with missing data. The algorithm is shown in its most basic form in Algorithm 16, but the usual tricks such as averaging and margin loss can be applied (Yu and Joachims, 2009). Aside from semantic parsing, the latent variable perceptron has been used in tasks such as machine translation (Liang et al., 2006) and named entity recognition (Sun et al., 2009). In **latent conditional random fields**, we use the full expectations rather than maximizing over the hidden variable. This model has also been employed in a range of problems beyond semantic parsing, including parse reranking (Koo and Collins, 2005) and gesture recognition (Quattoni et al., 2007).

12.4.3 Learning from denotations

Logical forms are easier to obtain than complete derivations, but the annotation of logical forms still requires considerable expertise. However, it is relatively easy to obtain denotations for many natural language sentences. For example, in the geography domain, the denotation of a question would be its answer (Clarke et al., 2010; Liang et al., 2013):

Text :What states border Georgia?
Logical form : $\lambda x.\text{STATE}(x) \wedge \text{BORDER}(x, \text{GEORGIA})$
Denotation :{Alabama, Florida, North Carolina,
 South Carolina, Tennessee}

Similarly, in a robotic control setting, the denotation of a command would be an action or sequence of actions (Artzi and Zettlemoyer, 2013). In both cases, the idea is to reward the semantic parser for choosing an analysis whose denotation is correct: the right answer to 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 $v_i(y) \in \{0, 1\}$ be a **validation function**, which assigns a

binary score indicating whether the denotation $\llbracket \mathbf{y} \rrbracket$ for the text $\mathbf{w}^{(i)}$ is correct. We can then learn by maximizing a conditional-likelihood objective,

$$\ell^{(i)}(\boldsymbol{\theta}) = \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times p(\mathbf{y} \mid \mathbf{w}; \boldsymbol{\theta}) \quad [12.29]$$

$$= \log \sum_{\mathbf{y}} v_i(\mathbf{y}) \times \sum_z p(\mathbf{y}, z \mid \mathbf{w}; \boldsymbol{\theta}), \quad [12.30]$$

which sums over all derivations z of all valid logical forms, $\{\mathbf{y} : v_i(\mathbf{y}) = 1\}$. This corresponds to the log-probability that the semantic parser produces a logical form with a valid denotation.

Differentiating with respect to $\boldsymbol{\theta}$, we obtain,

$$\frac{\partial \ell^{(i)}}{\partial \boldsymbol{\theta}} = \sum_{\mathbf{y}, z: v_i(\mathbf{y})=1} p(\mathbf{y}, z \mid \mathbf{w}) f(\mathbf{w}, z, \mathbf{y}) - \sum_{\mathbf{y}', z'} p(\mathbf{y}', z' \mid \mathbf{w}) f(\mathbf{w}, z', \mathbf{y}'), \quad [12.31]$$

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

6839 article by Liang and Potts (2015), tutorial slides and videos by Artzi and Zettlemoyer
 6840 (2013),¹² and the source code by Yoav Artzi¹³ and Percy Liang.¹⁴

6841 Exercises

- 6842 1. The **modus ponens** inference rule states that if we know $\phi \Rightarrow \psi$ and ϕ , then ψ must
 6843 be true. Justify this rule, using the definition of the \Rightarrow operator and some of the laws
 6844 provided in § 12.2.1, plus one additional identity: $\perp \vee \phi = \phi$.
- 6845 2. Convert the following examples into first-order logic, using the relations CAN-SLEEP,
 6846 MAKES-NOISE, and BROTHER.
 - 6847 • If Abigail makes noise, no one can sleep.
 - 6848 • If Abigail makes noise, someone cannot sleep.
 - 6849 • None of Abigail’s brothers can sleep.
 - 6850 • If one of Abigail’s brothers makes noise, Abigail cannot sleep.
- 6851 3. Extend the grammar fragment G_1 to include the ditransitive verb *teaches* and the
 6852 proper noun *Swahili*. Show how to derive the interpretation for the sentence *Alex*
 6853 *teaches Brit Swahili*, which should be $\text{TEACHES}(\text{ALEX}, \text{BRIT}, \text{SWAHILI})$. The grammar
 6854 need not be in Chomsky Normal Form. For the ditransitive verb, use NP_1 and NP_2
 6855 to indicate the two direct objects.
- 6856 4. Derive the semantic interpretation for the sentence *Alex likes every dog*, using gram-
 6857 mar fragment G_2 .
- 6858 5. Extend the grammar fragment G_2 to handle adjectives, so that the meaning of *an
 6859 angry dog* is $\lambda P. \exists x \text{DOG}(x) \wedge \text{ANGRY}(x) \wedge P(x)$. Specifically, you should supply the
 6860 lexical entry for the adjective *angry*, and you should specify the syntactic-semantic
 6861 productions $\text{NP} \rightarrow \text{DET } \text{NOM}$, $\text{NOM} \rightarrow \text{JJ } \text{NOM}$, and $\text{NOM} \rightarrow \text{NN}$.
- 6862 6. Extend your answer to the previous question to cover copula constructions with
 6863 predicative adjectives, such as *Alex is angry*. The interpretation should be $\text{ANGRY}(\text{ALEX})$.
 6864 You should add a verb phrase production $\text{VP} \rightarrow V_{\text{cop}} \text{JJ}$, and a terminal production
 6865 $V_{\text{cop}} \rightarrow \text{is}$. Show why your grammar extensions result in the correct interpretation.
- 6866 7. In Figure 12.5 and Figure 12.6, we treat the plurals *shoots* and *leaves* as entities. Revise
 6867 G_2 so that the interpretation of *Alex eats leaves* is $\forall x. (\text{LEAF}(x) \Rightarrow \text{EATS}(\text{ALEX}, x))$, and
 6868 show the resulting perceptron update.

¹²Videos are currently available at <http://yoavartzi.com/tutorial/>

¹³<http://yoavartzi.com/spf>

¹⁴<https://github.com/percyliang/sempre>

8. Statements like *every student eats a pizza* have two possible interpretations, depending on quantifier scope:

$$\forall x \exists y \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.32]$$

$$\exists y \forall x \text{PIZZA}(y) \wedge (\text{STUDENT}(x) \Rightarrow \text{EATS}(x, y)) \quad [12.33]$$

- 6869 a) Explain why these interpretations really are different.
- 6870 b) Which is generated by grammar G_2 ? Note that you may have to manipulate
6871 the logical form to exactly align with the grammar.
- 6872 9. *Modify G_2 so that produces the second interpretation in the previous problem.
6873 **Hint:** one possible solution involves changing the semantics of the sentence pro-
6874 duction and one other production.
- 6875 10. In the GeoQuery domain, give a natural language query that has multiple plausible
6876 semantic interpretations with the same denotation. List both interpretaions and the
6877 denotation.
- 6878 **Hint:** There are many ways to do this, but one approach involves using toponyms
6879 (place names) that could plausibly map to several different entities in the model.

6880

Chapter 13

6881

Predicate-argument semantics

6882 This chapter considers more “lightweight” semantic representations, which discard some
6883 aspects of first-order logic, but focus on predicate-argument structures. Let’s begin by
6884 thinking about the semantics of events, with a simple example:

6885 (13.1) Asha gives Boyang a book.

6886 A first-order logical representation of this sentence is,

$$\exists x. \text{BOOK}(x) \wedge \text{GIVE}(\text{ASHA}, \text{BOYANG}, x) \quad [13.1]$$

6887 In this representation, we define variable x for the book, and we link the strings *Asha* and
6888 *Boyang* to entities ASHA and BOYANG. Because the action of giving involves a giver, a
6889 recipient, and a gift, the predicate GIVE must take three arguments.

6890 Now suppose we have additional information about the event:

6891 (13.2) Yesterday, Asha reluctantly gave Boyang a book.

6892 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 ,

$$\begin{aligned} \exists e, x. & \text{GIVE-EVENT}(e) \wedge \text{GIVER}(e, \text{ASHA}) \wedge \text{GIFT}(e, x) \wedge \text{BOOK}(e, x) \wedge \text{RECIPIENT}(e, \text{BOYANG}) \\ & \wedge \text{TIME}(e, \text{YESTERDAY}) \wedge \text{MANNER}(e, \text{RELUCTANTLY}) \end{aligned}$$

6893 In this way, each argument of the event — the giver, the recipient, the gift — can be rep-
 6894 resented with a relation of its own, linking the argument to the event e . The expression
 6895 GIVER(e , ASHA) says that ASHA plays the **role** of GIVER in the event. This reformulation
 6896 handles the problem of optional information such as the time or manner of the event,
 6897 which are called **adjuncts**. Unlike arguments, adjuncts are not a mandatory part of the
 6898 relation, but under this representation, they can be expressed with additional logical rela-
 6899 tions that are conjoined to the semantic interpretation of the sentence.¹

6900 The event semantic representation can be applied to nested clauses, e.g.,

6901 (13.3) Chris sees Asha pay Boyang.

This is done by using the event variable as an argument:

$$\begin{aligned} \exists e_1 \exists e_2 \text{SEE-EVENT}(e_1) \wedge \text{SEER}(e_1, \text{CHRIS}) \wedge \text{SIGHT}(e_1, e_2) \\ \wedge \text{PAY-EVENT}(e_2) \wedge \text{PAYER}(e_2, \text{ASHA}) \wedge \text{PAYEE}(e_2, \text{BOYANG}) \end{aligned} \quad [13.3]$$

6902 As with first-order logic, the goal of event semantics is to provide a representation that
 6903 generalizes over many surface forms. Consider the following paraphrases of (13.1):

- 6904 (13.4) Asha gives a book to Boyang.
- 6905 (13.5) A book is given to Boyang by Asha.
- 6906 (13.6) A book is given by Asha to Boyang.
- 6907 (13.7) The gift of a book from Asha to Boyang ...

6908 All have the same event semantic meaning as Equation 13.1, but the ways in which the
 6909 meaning can be expressed are diverse. The final example does not even include a verb:
 6910 events are often introduced by verbs, but as shown by (13.7), the noun *gift* can introduce
 6911 the same predicate, with the same accompanying arguments.

6912 **Semantic role labeling** (SRL) is a relaxed form of semantic parsing, in which each
 6913 semantic role is filled by a set of tokens from the text itself. This is sometimes called
 6914 “shallow semantics” because, unlike model-theoretic semantic parsing, role fillers need
 6915 not be symbolic expressions with denotations in some world model. A semantic role
 6916 labeling system is required to identify all predicates, and then specify the spans of text
 6917 that fill each role. To give a sense of the task, here is a more complicated example:

- 6918 (13.8) Boyang wants Asha to give him a linguistics book.

¹This representation is often called **Neo-Davidsonian event semantics**. The use of existentially-quantified event variables was proposed by Davidson (1967) to handle the issue of optional adjuncts. In Neo-Davidsonian semantics, this treatment of adjuncts is extended to mandatory arguments as well (e.g., Parsons, 1990).

6919 In this example, there are two predicates, expressed by the verbs *want* and *give*. Thus, a
 6920 semantic role labeler might return the following output:

- 6921 • (PREDICATE : *wants*, WANTED : *Boyang*, DESIRE : *Asha to give him a linguistics book*)
 6922 • (PREDICATE : *give*, GIVER : *Asha*, RECIPIENT : *him*, GIFT : *a linguistics book*)

6923 *Boyang* and *him* may refer to the same person, but the semantic role labeling is not re-
 6924 quired to resolve this reference. Other predicate-argument representations, such as **Ab-**
 6925 **stract Meaning Representation (AMR)**, do require reference resolution. We will return to
 6926 AMR in § 13.3, but first, let us further consider the definition of semantic roles.

6927 **13.1 Semantic roles**

6928 In event semantics, it is necessary to specify a number of additional logical relations to
 6929 link arguments to events: GIVER, RECIPIENT, SEER, SIGHT, etc. Indeed, every predicate re-
 6930 quires a set of logical relations to express its own arguments. In contrast, adjuncts such as
 6931 TIME and MANNER are shared across many types of events. A natural question is whether
 6932 it is possible to treat mandatory arguments more like adjuncts, by identifying a set of
 6933 generic argument types that are shared across many event predicates. This can be further
 6934 motivated by examples involving related verbs:

- 6935 (13.9) Asha gave Boyang a book.
 6936 (13.10) Asha loaned Boyang a book.
 6937 (13.11) Asha taught Boyang a lesson.
 6938 (13.12) Asha gave Boyang a lesson.

6939 The respective roles of Asha, Boyang, and the book are nearly identical across the first
 6940 two examples. The third example is slightly different, but the fourth example shows that
 6941 the roles of GIVER and TEACHER can be viewed as related.

6942 One way to think about the relationship between roles such as GIVER and TEACHER is
 6943 by enumerating the set of properties that an entity typically possesses when it fulfills these
 6944 roles: givers and teachers are usually **animate** (they are alive and sentient) and **volitional**
 6945 (they choose to enter into the action).² In contrast, the thing that gets loaned or taught is
 6946 usually not animate or volitional; furthermore, it is unchanged by the event.

6947 Building on these ideas, **thematic roles** generalize across predicates by leveraging the
 6948 shared semantic properties of typical role fillers (Fillmore, 1968). For example, in exam-
 6949 ples (13.9-13.12), Asha plays a similar role in all four sentences, which we will call the

²There are always exceptions. For example, in the sentence *The C programming language has taught me a lot about perseverance*, the “teacher” is the *The C programming language*, which is presumably not animate or volitional.

| | | | | |
|-----------------|---------------|---------------|-----------------------|-------------------|
| | <i>Asha</i> | <i>gave</i> | <i>Boyang</i> | <i>a book</i> |
| VerbNet | AGENT | | RECIPIENT | THEME |
| PropBank | ARG0: giver | | ARG2: entity given to | ARG1: thing given |
| FrameNet | DONOR | | RECIPIENT | THEME |
| | <i>Asha</i> | <i>taught</i> | <i>Boyang</i> | <i>algebra</i> |
| VerbNet | AGENT | | RECIPIENT | TOPIC |
| PropBank | ARG0: teacher | | ARG2: student | ARG1: subject |
| FrameNet | TEACHER | | STUDENT | SUBJECT |

Figure 13.1: Example semantic annotations according to VerbNet, PropBank, and FrameNet

6950 **agent.** This reflects several shared semantic properties: she is the one who is actively and
 6951 intentionally performing the action, while Boyang is a more passive participant; the book
 6952 and the lesson would play a different role, as non-animate participants in the event.

6953 Example annotations from three well known systems are shown in Figure 13.1. We
 6954 will now discuss these systems in more detail.

6955 13.1.1 VerbNet

6956 **VerbNet** (Kipper-Schuler, 2005) is a lexicon of verbs, and it includes thirty “core” thematic
 6957 roles played by arguments to these verbs. Here are some example roles, accompanied by
 6958 their definitions from the VerbNet Guidelines.³

- 6959 • AGENT: “ACTOR in an event who initiates and carries out the event intentionally or
 6960 consciously, and who exists independently of the event.”
- 6961 • PATIENT: “UNDERGOER in an event that experiences a change of state, location or
 6962 condition, that is causally involved or directly affected by other participants, and
 6963 exists independently of the event.”
- 6964 • RECIPIENT: “DESTINATION that is animate”
- 6965 • THEME: “UNDERGOER that is central to an event or state that does not have control
 6966 over the way the event occurs, is not structurally changed by the event, and/or is
 6967 characterized as being in a certain position or condition throughout the state.”
- 6968 • TOPIC: “THEME characterized by information content transferred to another partic-
 6969 ipant.”

³http://verbs.colorado.edu/verb-index/VerbNet_Guidelines.pdf

6970 VerbNet roles are organized in a hierarchy, so that a TOPIC is a type of THEME, which in
 6971 turn is a type of UNDERGOER, which is a type of PARTICIPANT, the top-level category.

6972 In addition, VerbNet organizes verb senses into a class hierarchy, in which verb senses
 6973 that have similar meanings are grouped together. Recall from § 4.2 that multiple meanings
 6974 of the same word are called **senses**, and that WordNet identifies senses for many English
 6975 words. VerbNet builds on WordNet, so that verb classes are identified by the WordNet
 6976 senses of the verbs that they contain. For example, the verb class give-13.1 includes
 6977 the first WordNet sense of *loan* and the second WordNet sense of *lend*.

6978 Each VerbNet class or subclass takes a set of thematic roles. For example, give-13.1
 6979 takes arguments with the thematic roles of AGENT, THEME, and RECIPIENT;⁴ the pred-
 6980 icate TEACH takes arguments with the thematic roles AGENT, TOPIC, RECIPIENT, and
 6981 SOURCE.⁵ So according to VerbNet, *Asha* and *Boyang* play the roles of AGENT and RECIP-
 6982 IENT in the sentences,

6983 (13.13) Asha gave Boyang a book.

6984 (13.14) Asha taught Boyang algebra.

6985 The *book* and *algebra* are both THEMES, but *algebra* is a subcategory of THEME — a TOPIC
 6986 — because it consists of information content that is given to the receiver.

6987 13.1.2 Proto-roles and PropBank

6988 Detailed thematic role inventories of the sort used in VerbNet are not universally accepted.
 6989 For example, Dowty (1991, pp. 547) notes that “Linguists have often found it hard to agree
 6990 on, and to motivate, the location of the boundary between role types.” He argues that a
 6991 solid distinction can be identified between just two **proto-roles**:

6992 **Proto-Agent.** Characterized by volitional involvement in the event or state; sentience
 6993 and/or perception; causing an event or change of state in another participant; move-
 6994 ment; exists independently of the event.

6995 **Proto-Patient.** Undergoes change of state; causally affected by another participant; sta-
 6996 tionary relative to the movement of another participant; does not exist indepen-
 6997 dently of the event.⁶

⁴<https://verbs.colorado.edu/verb-index/vn/give-13.1.php>

⁵https://verbs.colorado.edu/verb-index/vn/transfer_mesg-37.1.1.php

⁶Reisinger et al. (2015) ask crowd workers to annotate these properties directly, finding that annotators tend to agree on the properties of each argument. They also find that in English, arguments having more proto-agent properties tend to appear in subject position, while arguments with more proto-patient properties appear in object position.

6998 In the examples in Figure 13.1, Asha has most of the proto-agent properties: in giving
 6999 the book to Boyang, she is acting volitionally (as opposed to *Boyang got a book from Asha*, in
 7000 which it is not clear whether Asha gave up the book willingly); she is sentient; she causes a
 7001 change of state in Boyang; she exists independently of the event. Boyang has some proto-
 7002 agent properties: he is sentient and exists independently of the event. But he also has
 7003 some proto-patient properties: he is the one who is causally affected and who undergoes
 7004 change of state. The book that Asha gives Boyang has even fewer of the proto-agent
 7005 properties: it is not volitional or sentient, and it has no causal role. But it also lacks many
 7006 of the proto-patient properties: it does not undergo change of state, exists independently
 7007 of the event, and is not stationary.

7008 The **Proposition Bank**, or PropBank (Palmer et al., 2005), builds on this basic agent-
 7009 patient distinction, as a middle ground between generic thematic roles and roles that are
 7010 specific to each predicate. Each verb is linked to a list of numbered arguments, with ARG0
 7011 as the proto-agent and ARG1 as the proto-patient. Additional numbered arguments are
 7012 verb-specific. For example, for the predicate TEACH,⁷ the arguments are:

- 7013 • ARG0: the teacher
- 7014 • ARG1: the subject
- 7015 • ARG2: the student(s)

7016 Verbs may have any number of arguments: for example, WANT and GET have five, while
 7017 EAT has only ARG0 and ARG1. In addition to the semantic arguments found in the frame
 7018 files, roughly a dozen general-purpose adjuncts may be used in combination with any
 7019 verb. These are shown in Table 13.1.

7020 PropBank-style semantic role labeling is annotated over the entire Penn Treebank. This
 7021 annotation includes the sense of each verbal predicate, as well as the argument spans.

7022 13.1.3 FrameNet

7023 Semantic **frames** are descriptions of situations or events. Frames may be *evoked* by one
 7024 of their **lexical units** (often a verb, but not always), and they include some number of
 7025 **frame elements**, which are like roles (Fillmore, 1976). For example, the act of teaching
 7026 is a frame, and can be evoked by the verb *taught*; the associated frame elements include
 7027 the teacher, the student(s), and the subject being taught. Frame semantics has played a
 7028 significant role in the history of artificial intelligence, in the work of Minsky (1974) and
 7029 Schank and Abelson (1977). In natural language processing, the theory of frame semantics
 7030 has been implemented in **FrameNet** (Fillmore and Baker, 2009), which consists of a lexicon

⁷<http://verbs.colorado.edu/propbank/framesets-english-aliases/teach.html>

| | | |
|-----|----------------------|--|
| TMP | time | <i>Boyang ate a bagel</i> [AM-TMP <i>yesterday</i>]. |
| LOC | location | <i>Asha studies in</i> [AM-LOC <i>Stuttgart</i>] |
| MOD | modal verb | <i>Asha</i> [AM-MOD <i>will</i>] <i>study in Stuttgart</i> |
| ADV | general purpose | [AM-ADV <i>Luckily</i>], <i>Asha knew algebra</i> . |
| MNR | manner | <i>Asha ate</i> [AM-MNR <i>aggressively</i>]. |
| DIS | discourse connective | [AM-DIS <i>However</i>], <i>Asha prefers algebra</i> . |
| PRP | purpose | <i>Barry studied</i> [AM-PRP <i>to pass the bar</i>]. |
| DIR | direction | <i>Workers dumped burlap sacks</i> [AM-DIR <i>into a bin</i>]. |
| NEG | negation | <i>Asha does</i> [AM-NEG <i>not</i>] <i>speak Albanian</i> . |
| EXT | extent | <i>Prices increased</i> [AM-EXT <i>4%</i>]. |
| CAU | cause | <i>Boyang returned the book</i> [AM-CAU <i>because it was overdue</i>]. |

Table 13.1: PropBank adjuncts (Palmer et al., 2005), sorted by frequency in the corpus

7031 of roughly 1000 frames, and a corpus of more than 200,000 “exemplar sentences,” in which
 7032 the frames and their elements are annotated.⁸

7033 Rather than seeking to link semantic roles such as TEACHER and GIVER into the-
 7034 matic roles such as AGENT, FrameNet aggressively groups verbs into frames, and links
 7035 semantically-related roles across frames. For example, the following two sentences would
 7036 be annotated identically in FrameNet:

7037 (13.15) Asha taught Boyang algebra.

7038 (13.16) Boyang learned algebra from Asha.

7039 This is because *teach* and *learn* are both lexical units in the EDUCATION-TEACHING frame.
 7040 Furthermore, roles can be shared even when the frames are distinct, as in the following
 7041 two examples:

7042 (13.17) Asha gave Boyang a book.

7043 (13.18) Boyang got a book from Asha.

7044 The GIVING and GETTING frames both have RECIPIENT and THEME elements, so Boyang
 7045 and the book would play the same role. Asha’s role is different: she is the DONOR in the
 7046 GIVING frame, and the SOURCE in the GETTING frame. FrameNet makes extensive use of
 7047 multiple inheritance to share information across frames and frame elements: for example,
 7048 the COMMERCE-SELL and LENDING frames inherit from GIVING frame.

⁸Current details and data can be found at <https://framenet.icsi.berkeley.edu/>

7049 **13.2 Semantic role labeling**

7050 The task of semantic role labeling is to identify the parts of the sentence comprising the
 7051 semantic roles. In English, this task is typically performed on the PropBank corpus, with
 7052 the goal of producing outputs in the following form:

7053 (13.19) [ARG0 Asha] [GIVE.01 gave] [ARG2 Boyang's mom] [ARG1 a book] [AM-TMP yesterday].

7054 Note that a single sentence may have multiple verbs, and therefore a given word may be
 7055 part of multiple role-fillers:

7056 (13.20) [ARG0 Asha] [WANT.01 wanted]
 Asha wanted

7057 [ARG1 Boyang to give her the book].
 [ARG0 Boyang] [GIVE.01 to give] [ARG2 her] [ARG1 the book].

7058 **13.2.1 Semantic role labeling as classification**

7059 PropBank is annotated on the Penn Treebank, and annotators used phrasal constituents
 7060 (\S 9.2.2) to fill the roles. PropBank semantic role labeling can be viewed as the task of as-
 7061 signing to each phrase a label from the set $\mathcal{R} = \{\emptyset, \text{PRED}, \text{ARG0}, \text{ARG1}, \text{ARG2}, \dots, \text{AM-LOC}, \text{AM-TMP}, \dots\}$,
 7062 with respect to each predicate. If we treat semantic role labeling as a classification prob-
 7063 lem, we obtain the following functional form:

$$\hat{y}_{(i,j)} = \underset{y}{\operatorname{argmax}} \psi(\mathbf{w}, y, i, j, \rho, \tau), \quad [13.4]$$

7064 where,

- 7065 • (i, j) indicates the span of a phrasal constituent $(w_{i+1}, w_{i+2}, \dots, w_j)$;⁹
- 7066 • \mathbf{w} represents the sentence as a sequence of tokens;
- 7067 • ρ is the index of the predicate verb in \mathbf{w} ;
- 7068 • τ is the structure of the phrasal constituent parse of \mathbf{w} .

7069 Early work on semantic role labeling focused on discriminative feature-based models,
 7070 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-
 7071 inal paper on FrameNet semantic role labeling (Gildea and Jurafsky, 2002). By 2005 there

⁹PropBank roles can also be filled by **split constituents**, which are discontinuous spans of text. This situation most frequently in reported speech, e.g. [ARG1 *By addressing these problems*], *Mr. Maxwell said*, [ARG1 *the new funds have become extremely attractive.*] (example adapted from Palmer et al., 2005). This issue is typically addressed by defining “continuation arguments”, e.g. C-ARG1, which refers to the continuation of ARG1 after the split.

| | |
|------------------------------------|--|
| Predicate lemma and POS tag | The lemma of the predicate verb and its part-of-speech tag |
| Voice | Whether the predicate is in active or passive voice, as determined by a set of syntactic patterns for identifying passive voice constructions |
| Phrase type | The constituent phrase type for the proposed argument in the parse tree, e.g. NP, PP |
| Headword and POS tag | The head word of the proposed argument and its POS tag, identified using the Collins (1997) rules |
| Position | Whether the proposed argument comes before or after the predicate in the sentence |
| Syntactic path | The set of steps on the parse tree from the proposed argument to the predicate (described in detail in the text) |
| Subcategorization | The syntactic production from the first branching node above the predicate. For example, in Figure 13.2, the subcategorization feature around <i>taught</i> would be VP → VBD NP PP. |

Table 13.2: Features used in semantic role labeling by Gildea and Jurafsky (2002).

were several systems for PropBank semantic role labeling, and their approaches and feature sets are summarized by Carreras and Márquez (2005). Typical features include: the phrase type, head word, part-of-speech, boundaries, and neighbors of the proposed argument $w_{i+1:j}$; the word, lemma, part-of-speech, and voice of the verb w_ρ (active or passive), as well as features relating to its frameset; the distance and path between the verb and the proposed argument. In this way, semantic role labeling systems are high-level “consumers” in the NLP stack, using features produced from lower-level components such as part-of-speech taggers and parsers. More comprehensive feature sets are enumerated by Das et al. (2014) and Täckström et al. (2015).

A particularly powerful class of features relate to the **syntactic path** between the argument and the predicate. These features capture the sequence of moves required to get from the argument to the verb by traversing the phrasal constituent parse of the sentence. The idea of these features is to capture syntactic regularities in how various arguments are realized. Syntactic path features are best illustrated by example, using the parse tree in Figure 13.2:

- The path from *Asha* to the verb *taught* is NNP↑NP↑S↓VP↓VBD. The first part of the path, NNP↑NP↑S, means that we must travel up the parse tree from the NNP tag (proper noun) to the S (sentence) constituent. The second part of the path, S↓VP↓VBD, means that we reach the verb by producing a VP (verb phrase) from

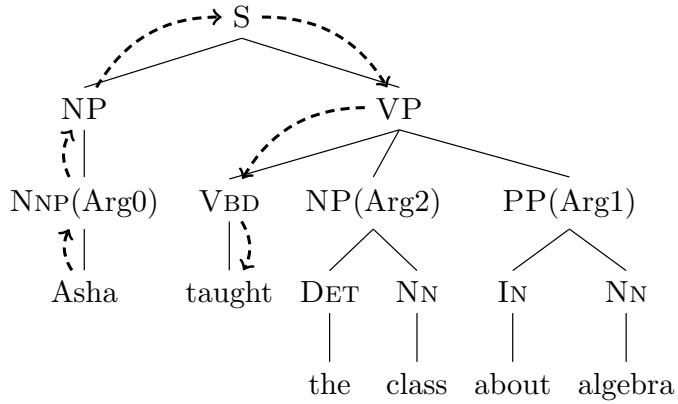


Figure 13.2: Semantic role labeling on the phrase-structure parse tree for a sentence. The dashed line indicates the syntactic path from *Asha* to the predicate verb *taught*.

7091 the S constituent, and then by producing a VBD (past tense verb). This feature is
 7092 consistent with *Asha* being in subject position, since the path includes the sentence
 7093 root S.

- 7094 • The path from *the class* to *taught* is NP↑VP↓VBD. This is consistent with *the class*
 7095 being in object position, since the path passes through the VP node that dominates
 7096 the verb *taught*.

7097 Because there are many possible path features, it can also be helpful to look at smaller
 7098 parts: for example, the upward and downward parts can be treated as separate features;
 7099 another feature might consider whether S appears anywhere in the path.

7100 Rather than using the constituent parse, it is also possible to build features from the **de-**
 7101 **pendency path** (see § 11.4) between the head word of each argument and the verb (Prad-
 7102 han et al., 2005). Using the Universal Dependency part-of-speech tagset and dependency
 7103 relations (Nivre et al., 2016), the dependency path from *Asha* to *taught* is PROPN $\xleftarrow[\text{NSUBJ}]{} \text{VERB}$,
 7104 because *taught* is the head of a relation of type $\xleftarrow[\text{NSUBJ}]{} \text{with}$ *Asha*. Similarly, the dependency
 7105 path from *class* to *taught* is NOUN $\xleftarrow[\text{DOBJ}]{} \text{VERB}$, because *class* heads the noun phrase that is a
 7106 direct object of *taught*. A more interesting example is *Asha wanted to teach the class*, where
 7107 the path from *Asha* to *teach* is PROPN $\xleftarrow[\text{NSUBJ}]{} \text{VERB} \rightarrow[\text{XCOMP}] \text{VERB}$. The right-facing arrow in sec-
 7108 ond relation indicates that *wanted* is the head of its XCOMP relation with *teach*.

7109 **13.2.2 Semantic role labeling as constrained optimization**

7110 A potential problem with treating SRL as a classification problem is that there are a num-
 7111 ber of sentence-level **constraints**, which a classifier might violate.

- 7112 • For a given verb, there can be only one argument of each type (ARG0, ARG1, etc.)
 7113 • Arguments cannot overlap. This problem arises when we are labeling the phrases
 7114 in a constituent parse tree, as shown in Figure 13.2: if we label the PP *about algebra*
 7115 as an argument or adjunct, then its children *about* and *algebra* must be labeled as \emptyset .
 7116 The same constraint also applies to the syntactic ancestors of this phrase.

7117 These constraints introduce dependencies across labeling decisions. In structure pre-
 7118 diction problems such as sequence labeling and parsing, such dependencies are usually
 7119 handled by defining a scoring over the entire structure, \mathbf{y} . Efficient inference requires
 7120 that the global score decomposes into local parts: for example, in sequence labeling, the
 7121 scoring function decomposes into scores of pairs of adjacent tags, permitting the applica-
 7122 tion of the Viterbi algorithm for inference. But the constraints that arise in semantic role
 7123 labeling are less amenable to local decomposition.¹⁰ We therefore consider **constrained**
 7124 **optimization** as an alternative solution.

Let the set $\mathcal{C}(\tau)$ refer to all labelings that obey the constraints introduced by the parse τ . The semantic role labeling problem can be reformulated as a constrained optimization over $\mathbf{y} \in \mathcal{C}(\tau)$,

$$\begin{aligned} \max_{\mathbf{y}} \quad & \sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) \\ \text{s.t. } \quad & \mathbf{y} \in \mathcal{C}(\tau). \end{aligned} \quad [13.5]$$

7125 In this formulation, the objective (shown on the first line) is a separable function of each
 7126 individual labeling decision, but the constraints (shown on the second line) apply to the
 7127 overall labeling. The sum $\sum_{(i,j) \in \tau}$ indicates that we are summing over all constituent
 7128 spans in the parse τ . The expression s.t. in the second line means that we maximize the
 7129 objective *subject to* the constraint $\mathbf{y} \in \mathcal{C}(\tau)$.

7130 A number of practical algorithms exist for restricted forms of constrained optimiza-
 7131 tion. One such restricted form is **integer linear programming**, in which the objective and
 7132 constraints are linear functions of integer variables. To formulate SRL as an integer linear
 7133 program, we begin by rewriting the labels as a set of binary variables $\mathbf{z} = \{z_{i,j,r}\}$ (Pun-
 7134 yakanok et al., 2008),

$$z_{i,j,r} = \begin{cases} 1, & y_{i,j} = r \\ 0, & \text{otherwise,} \end{cases} \quad [13.6]$$

¹⁰Dynamic programming solutions have been proposed by Tromble and Eisner (2006) and Täckström et al. (2015), but they involve creating a trellis structure whose size is exponential in the number of labels.

7135 where $r \in \mathcal{R}$ is a label in the set $\{\text{ARG0}, \text{ARG1}, \dots, \text{AM-LOC}, \dots, \emptyset\}$. Thus, the variables
 7136 \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} .

$$\sum_{(i,j) \in \tau} \psi(\mathbf{w}, y_{i,j}, i, j, \rho, \tau) = \sum_{i,j,r} \psi(\mathbf{w}, r, i, j, \rho, \tau) \times z_{i,j,r}, \quad [13.7]$$

7137 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,

$$\forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.8]$$

7138 Recall that $z_{i,j,r} = 1$ iff the span (i, j) has label r ; this constraint says that for each possible
 7139 label $r \neq \emptyset$, there can be at most one (i, j) such that $z_{i,j,r} = 1$. Rewriting this constraint
 7140 can be written in the form $\mathbf{A}\mathbf{z} \leq \mathbf{b}$, as you will find if you complete the exercises at the
 7141 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:

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1, \quad [13.9]$$

7142 where $o((i, j), (i, j)) = 1$.

In summary, the semantic role labeling problem can thus be rewritten as the following integer linear program,

$$\max_{\mathbf{z} \in \{0,1\}^{|\tau|}} \quad \sum_{(i,j) \in \tau} \sum_{r \in \mathcal{R}} z_{i,j,r} \psi_{i,j,r} \quad [13.10]$$

$$s.t. \quad \forall r \neq \emptyset, \quad \sum_{(i,j) \in \tau} z_{i,j,r} \leq 1. \quad [13.11]$$

$$\forall (i, j) \in \tau, \quad \sum_{(i', j') \in \tau} \sum_{r \neq \emptyset} o((i, j), (i', j')) \times z_{i',j',r} \leq 1. \quad [13.12]$$

7143 **Learning with constraints** Learning can be performed in the context of constrained op-
 7144 timization using the usual perceptron or large-margin classification updates. Because
 7145 constrained inference is generally more time-consuming, a key question is whether it is
 7146 necessary to apply the constraints during learning. Chang et al. (2008) find that better per-
 7147 formance can be obtained by learning *without* constraints, and then applying constraints
 7148 only when using the trained model to predict semantic roles for unseen data.

7149 **How important are the constraints?** Das et al. (2014) find that an unconstrained, classification-
 7150 based method performs nearly as well as constrained optimization for FrameNet parsing:
 7151 while it commits many violations of the “no-overlap” constraint, the overall F_1 score is
 7152 less than one point worse than the score at the constrained optimum. Similar results
 7153 were obtained for PropBank semantic role labeling by Punyakanok et al. (2008). He et al.
 7154 (2017) find that constrained inference makes a bigger impact if the constraints are based
 7155 on manually-labeled “gold” syntactic parses. This implies that errors from the syntac-
 7156 tic parser may limit the effectiveness of the constraints. Punyakanok et al. (2008) hedge
 7157 against parser error by including constituents from several different parsers; any con-
 7158 stituent can be selected from any parse, and additional constraints ensure that overlap-
 7159 ping constituents are not selected.

7160 **Implementation** Integer linear programming solvers such as `glpk`,¹¹ `cplex`,¹² and `Gurobi`¹³
 7161 allow inequality constraints to be expressed directly in the problem definition, rather than
 7162 in the matrix form $\mathbf{A}z \leq \mathbf{b}$. The time complexity of integer linear programming is theoreti-
 7163 cally exponential in the number of variables $|z|$, but in practice these off-the-shelf solvers
 7164 obtain good solutions efficiently. Using a standard desktop computer, Das et al. (2014)
 7165 report that the `cplex` solver requires 43 seconds to perform inference on the FrameNet
 7166 test set, which contains 4,458 predicates.

7167 Recent work has shown that many constrained optimization problems in natural lan-
 7168 guage processing can be solved in a highly parallelized fashion, using optimization tech-
 7169 niques such as **dual decomposition**, which are capable of exploiting the underlying prob-
 7170 lem structure (Rush et al., 2010). Das et al. (2014) apply this technique to FrameNet se-
 7171 mantic role labeling, obtaining an order-of-magnitude speedup over `cplex`.

7172 13.2.3 Neural semantic role labeling

7173 Neural network approaches to SRL have tended to treat it as a sequence labeling task,
 7174 using a labeling scheme such as the **BIO notation**, which we previously saw in named
 7175 entity recognition (§ 8.3). In this notation, the first token in a span of type ARG1 is labeled

¹¹<https://www.gnu.org/software/glpk/>

¹²<https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

¹³<http://www.gurobi.com/>

7176 B-ARG1; all remaining tokens in the span are *inside*, and are therefore labeled I-ARG1.
 7177 Tokens outside any argument are labeled O. For example:

- 7178 (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 (§ 7.6) 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 $\mathbf{Z}^{(K)} = [z_1^{(K)}, z_2^{(K)}, \dots, z_M^{(K)}]$. The “emission” score for each tag $Y_m = y$ is equal to the inner product $\theta_y \cdot z_m^{(K)}$, and there is also a transition score for each pair of adjacent tags. The complete model can be written,

$$\mathbf{Z}^{(1)} = \text{BiLSTM}(\mathbf{X}) \quad [13.13]$$

$$\mathbf{Z}^{(i)} = \text{BiLSTM}(\mathbf{Z}^{(i-1)}) \quad [13.14]$$

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\text{argmax}} \sum_{m=1}^M \Theta^{(y)} \mathbf{z}_m^{(K)} + \psi_{y_{m-1}, y_m}. \quad [13.15]$$

7179 Note that the final step maximizes over the entire labeling \mathbf{y} , and includes a score for
 7180 each tag transition ψ_{y_{m-1}, y_m} . This combination of LSTM and pairwise potentials on tags
 7181 is an example of an **LSTM-CRF**. The maximization over \mathbf{y} is performed by the Viterbi
 7182 algorithm.

7183 This model strongly outperformed alternative approaches at the time, including con-
 7184 strained decoding and convolutional neural networks.¹⁴ More recent work has combined
 7185 recurrent neural network models with constrained decoding, using the A^* search algo-
 7186 rithm to search over labelings that are feasible with respect to the constraints (He et al.,
 7187 2017). This yields small improvements over the method of Zhou and Xu (2015). He et al.
 7188 (2017) obtain larger improvements by creating an **ensemble** of SRL systems, each trained
 7189 on an 80% subsample of the corpus. The average prediction across this ensemble is more
 7190 robust than any individual model.

7191 13.3 Abstract Meaning Representation

7192 Semantic role labeling transforms the task of semantic parsing to a labeling task. Consider
 7193 the sentence,

¹⁴The successful application of **convolutional neural networks** to semantic role labeling by Collobert and Weston (2008) was an influential early result in the current wave of neural networks in natural language processing.

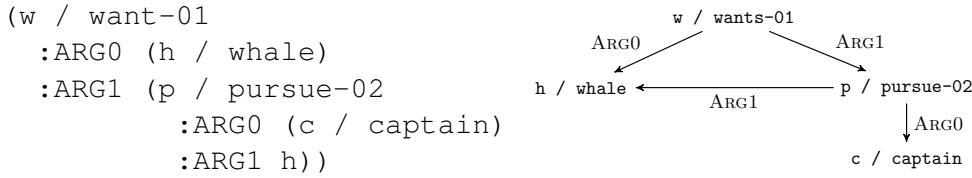


Figure 13.3: Two views of the AMR representation for the sentence *The whale wants the captain to pursue him.*

7194 (13.22) The whale wants the captain to pursue him.

7195 The PropBank semantic role labeling analysis is:

7196 • (PREDICATE : *wants*, ARG0 : *the whale*, ARG1 : *the captain to pursue him*)

7197 • (PREDICATE : *pursue*, ARG0 : *the captain*, ARG1 : *him*)

7198 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 7199 PENMAN notation (Matthiessen and Bateman, 1991), in which each set of parentheses 7200 introduces a variable. Each variable is an **instance** of a concept, which is indicated with the 7201 slash notation: for example, *w / want-01* indicates that the variable *w* is an instance of 7202 the concept *want-01*, which in turn refers to the PropBank frame for the first sense of the 7203 verb *want*; *pursue-02* refers to the second sense of *pursue*. Relations are introduced with 7204 colons: for example, *:ARG0 (c / captain)* indicates a relation of type ARG0 with the 7205 newly-introduced variable *c*. Variables can be reused, so that when the variable *h* ap- 7206 pears again as an argument to *p*, it is understood to refer to the same whale in both cases. 7207 This arrangement is indicated compactly in the graph structure on the right, with edges 7208 indicating concepts. 7209

7210 One way in which AMR differs from PropBank-style semantic role labeling is that it 7211 reifies each entity as a variable: for example, the *whale* in (13.22) is reified in the variable 7212 *h*, which is reused as ARG0 in its relationship with *w / want-01*, and as ARG1 in its 7213 relationship with *p / pursue-02*. Reifying entities as variables also makes it possible 7214 to represent the substructure of noun phrases more explicitly. For example, *Asha borrowed* 7215 *the algebra book* would be represented as:

7217 (b / borrow-01
 7218 :ARG0 (p / person
 7219 :name (n / name
 7220 :op1 "Asha"))

```

7221   :ARG1 (b2 / book
7222       :topic (a / algebra)))

```

7223 This indicates that the variable *p* is a person, whose name is the variable *n*; that name
 7224 has one token, the string *Asha*. Similarly, the variable *b2* is a book, and the *topic* of *b2*
 7225 is a variable *a* whose type is *algebra*. The relations *name* and *topic* are examples of
 7226 “non-core roles”, which are similar to adjunct modifiers in PropBank. However, AMR’s
 7227 inventory is more extensive, including more than 70 non-core roles, such as negation,
 7228 time, manner, frequency, and location. Lists and sequences — such as the list of tokens in
 7229 a name — are described using the roles *op1*, *op2*, etc.

7230 Another feature of AMR is that a semantic predicate can be introduced by any syntac-
 7231 tic element, as in the following examples from Banerescu et al. (2013):

- 7232 (13.23) The boy destroyed the room.
- 7233 (13.24) the destruction of the room by the boy ...
- 7234 (13.25) the boy’s destruction of the room ...

7235 All these examples have the same semantics in AMR,

```

7236 (d / destroy-01
7237   :ARG0 (b / boy)
7238   :ARG1 (r / room))

```

7239 The noun *destruction* is linked to the verb *destroy*, which is captured by the PropBank
 7240 frame *destroy-01*. This can happen with adjectives as well: in the phrase *the attractive*
 7241 *spy*, the adjective *attractive* is linked to the PropBank frame *attract-01*:

```

7242 (s / spy
7243   :ARG0-of (a / attract-01))

```

7244 In this example, *ARG0-of* is an **inverse relation**, indicating that *s* is the *ARG0* of the
 7245 predicate *a*. Inverse relations make it possible for all AMR parses to have a single root
 7246 concept.

7247 While AMR goes farther than semantic role labeling, it does not link semantically-
 7248 related frames such as *buy*/*sell* (as FrameNet does). AMR also does not handle quanti-
 7249 fication (as first-order predicate calculus does), and it makes no attempt to handle noun
 7250 number and verb tense (as PropBank does).

13.3.1 AMR Parsing

Abstract Meaning Representation is not a labeling of the original text — unlike PropBank semantic role labeling, and most of the other tagging and parsing tasks that we have encountered thus far. The AMR for a given sentence may include multiple concepts for single words in the sentence: as we have seen, the sentence *Asha likes algebra* contains both person and name concepts for the word *Asha*. Conversely, words in the sentence may not appear in the AMR: in *Boyang made a tour of campus*, the light verb *make* would not appear in the AMR, which would instead be rooted on the predicate *tour*. As a result, AMR is difficult to parse, and even evaluating AMR parsing involves considerable algorithmic complexity (Cai and Yates, 2013).

A further complexity is that AMR labeled datasets do not explicitly show the alignment between the AMR annotation and the words in the sentence. For example, the link between the word *wants* and the concept *want-01* is not annotated. To acquire training data for learning-based parsers, it is therefore necessary to first perform an alignment between the training sentences and their AMR parses. Flanigan et al. (2014) introduce a rule-based parser, which links text to concepts through a series of increasingly high-recall steps.

As with dependency parsing, AMR can be parsed by graph-based methods that explore the space of graph structures, or by incremental transition-based algorithms. One approach to graph-based AMR parsing is to first group adjacent tokens into local substructures, and then to search the space of graphs over these substructures (Flanigan et al., 2014). The identification of concept subgraphs can be formulated as a sequence labeling problem, and the subsequent graph search can be solved using integer linear programming (§ 13.2.2). Various transition-based parsing algorithms have been proposed. Wang et al. (2015) construct an AMR graph by incrementally modifying the syntactic dependency graph. At each step, the parser performs an action: for example, adding an AMR relation label to the current dependency edge, swapping the direction of a syntactic dependency edge, or cutting an edge and reattaching the orphaned subtree to a new parent.

Additional resources

Practical semantic role labeling was first made possible by the PropBank annotations on the Penn Treebank (Palmer et al., 2005). Abend and Rappoport (2017) survey several semantic representation schemes, including semantic role labeling and AMR. Other linguistic features of AMR are summarized in the original paper (Banarescu et al., 2013) and the tutorial slides by Schneider et al. (2015). Recent shared tasks have undertaken semantic dependency parsing, in which the goal is to identify semantic relationships between pairs of words (Oepen et al., 2014); see Ivanova et al. (2012) for an overview of connections between syntactic and semantic dependencies.

7288 **Exercises**

7289 1. Write out an event semantic representation for the following sentences. You may
 7290 make up your own predicates.

7291 (13.26) *Abigail shares with Max.*

7292 (13.27) *Abigail reluctantly shares a toy with Max.*

7293 (13.28) *Abigail hates to share with Max.*

7294 2. Find the PropBank framesets for *share* and *hate* at <http://verbs.colorado.edu/propbank/framesets-english-aliases/>, and rewrite your answers from the
 7295 previous question, using the thematic roles ARG0, ARG1, and ARG2.

7296 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
 7297 sure about the parse, you can try an online parser such as <http://nlp.stanford.edu:8080/parser/>.

7298 4. Compute the dependency path features for Abigail and Max in each of the example
 7299 sentences (13.26) and (13.28) in Question 1, with respect to the verb *share*. Again, if
 7300 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*
 7301 and *Max* is OBL according to the Universal Dependency treebank.

7302 5. PropBank semantic role labeling includes **reference arguments**, such as,

7303 (13.29) [AM-LOC *The bed*] on [R-AM-LOC *which*] I slept broke.¹⁵

7304 The label R-AM-LOC indicates that the word *which* is a reference to *The bed*, which
 7305 expresses the location of the event. Reference arguments must have referents: the
 7306 tag R-AM-LOC can appear only when AM-LOC also appears in the sentence. Show
 7307 how to express this as a linear constraint, specifically for the tag R-AM-LOC. Be sure
 7308 to correctly handle the case in which neither AM-LOC nor R-AM-LOC appear in the
 7309 sentence.

7310 6. Explain how to express the constraints on semantic role labeling in Equation 13.8
 7311 and Equation 13.9 in the general form $Az \geq b$.

7312 7. Produce the AMR annotations for the following examples:

7313 (13.30) *The girl likes the boy.*

¹⁵Example from 2013 NAACL tutorial slides by Shumin Wu

- 7318 (13.31) The girl was liked by the boy.
 7319 (13.32) Abigail likes Maxwell Aristotle.
 7320 (13.33) The spy likes the attractive boy.
 7321 (13.34) The girl doesn't like the boy.
 7322 (13.35) The girl likes her dog.

7323 For (13.32), recall that multi-token names are created using `op1`, `op2`, etc. You will
 7324 need to consult Banarescu et al. (2013) for (13.34), and Schneider et al. (2015) for
 7325 (13.35). You may assume that *her* refers to *the girl* in this example.

- 7326 8. In this problem, you will build a FrameNet sense classifier for the verb *can*, which
 7327 can evoke two frames: POSSIBILITY (can you order a salad with french fries?) and
 7328 CAPABILITY (can you eat a salad with chopsticks?).

7329 To build the dataset, access the FrameNet corpus in NLTK:

```
7330 import nltk
7331 nltk.download('framenet_v17')
7332 from nltk.corpus import framenet as fn
```

7333 Next, find instances in which the lexical unit `can.v` (the verb form of *can*) evokes a
 7334 frame. Do this by iterating over `fn.docs()`, and then over sentences, and then

```
7335 for doc in fn.docs():
7336     if 'sentence' in doc:
7337         for sent in doc['sentence']:
7338             for anno_set in sent['annotationSet']:
7339                 if 'luName' in anno_set and anno_set['luName'] == 'can.v':
7340                     pass # your code here
```

7341 Use the field `frameName` as a label, and build a set of features from the field `text`.
 7342 Train a classifier to try to accurately predict the `frameName`, disregarding cases
 7343 other than CAPABILITY and POSSIBILITY. Treat the first hundred instances as a training
 7344 set, and the remaining instances as the test set. Can you do better than a classifier
 7345 that simply selects the most common class?

- 7346 9. *Download the PropBank sample data, using NLTK (<http://www.nltk.org/howto/propbank.html>).

- 7348 a) Use a deep learning toolkit such as PyTorch to train a BiLSTM sequence labeling
 7349 model (§ 7.6) to identify words or phrases that are predicates, e.g., *we/O*
 7350 *took/B-PRED a/I-PRED walk/I-PRED together/O*. Your model should compute
 7351 the tag score from the BiLSTM hidden state $\psi(y_m) = \beta_y \cdot h_m$.
- 7352 b) Optionally, implement Viterbi to improve the predictions of the model in the
 7353 previous section.

7354 c) Try to identify ARG0 and ARG1 for each predicate. You should again use the
 7355 BiLSTM and BIO notation, but you may want to include the BiLSTM hidden
 7356 state at the location of the predicate in your prediction model, e.g., $\psi(y_m) =$
 7357 $\beta_y \cdot [\mathbf{h}_m; \mathbf{h}_{\hat{r}}]$, where \hat{r} is the predicted location of the (first word of the) predicate.

7358 10. Using an off-the-shelf PropBank SRL system,¹⁶ build a simplified question answer-
 7359 ing system in the style of Shen and Lapata (2007). Specifically, your system should
 7360 do the following:

- 7361 • For each document in a collection, it should apply the semantic role labeler,
 7362 and should store the output as a tuple.
- 7363 • For a question, your system should again apply the semantic role labeler. If
 7364 any of the roles are filled by a *wh*-pronoun, you should mark that role as the
 7365 expected answer phrase (EAP).
- 7366 • To answer the question, search for a stored tuple which matches the question as
 7367 well as possible (same predicate, no incompatible semantic roles, and as many
 7368 matching roles as possible). Align the EAP against its role filler in the stored
 7369 tuple, and return this as the answer.

7370 To evaluate your system, download a set of three news articles on the same topic,
 7371 and write down five factoid questions that should be answerable from the arti-
 7372 cles. See if your system can answer these questions correctly. (If this problem is
 7373 assigned to an entire class, you can build a large-scale test set and compare various
 7374 approaches.)

¹⁶At the time of writing, the following systems are available: SENNA (<http://ronan.collobert.com/senna/>), Illinois Semantic Role Labeler (https://cogcomp.cs.illinois.edu/page/software_view/SRL), and mate-tools (<https://code.google.com/archive/p/mate-tools/>).

7375 Chapter 14

7376 Distributional and distributed 7377 semantics

7378 A recurring theme in natural language processing is the complexity of the mapping from
7379 words to meaning. In chapter 4, we saw that a single word form, like *bank*, can have mul-
7380 tiple meanings; conversely, a single meaning may be created by multiple surface forms,
7381 a lexical semantic relationship known as **synonymy**. Despite this complex mapping be-
7382 tween words and meaning, natural language processing systems usually rely on words
7383 as the basic unit of analysis. This is especially true in semantics: the logical and frame
7384 semantic methods from the previous two chapters rely on hand-crafted lexicons that map
7385 from words to semantic predicates. But how can we analyze texts that contain words
7386 that we haven't seen before? This chapter describes methods that learn representations
7387 of word meaning by analyzing unlabeled data, vastly improving the generalizability of
7388 natural language processing systems. The theory that makes it possible to acquire mean-
7389 ingful representations from unlabeled data is the **distributional hypothesis**.

7390 14.1 The distributional hypothesis

7391 Here's a word you may not know: *tezgüino* (the example is from Lin, 1998). If you do not
7392 know the meaning of *tezgüino*, then you are in the same situation as a natural language
7393 processing system when it encounters a word that did not appear in its training data.
7394 Now suppose you see that *tezgüino* is used in the following contexts:

- 7395 (14.1) A bottle of _____ is on the table.
- 7396 (14.2) Everybody likes _____.
- 7397 (14.3) Don't have _____ before you drive.
- 7398 (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 | |

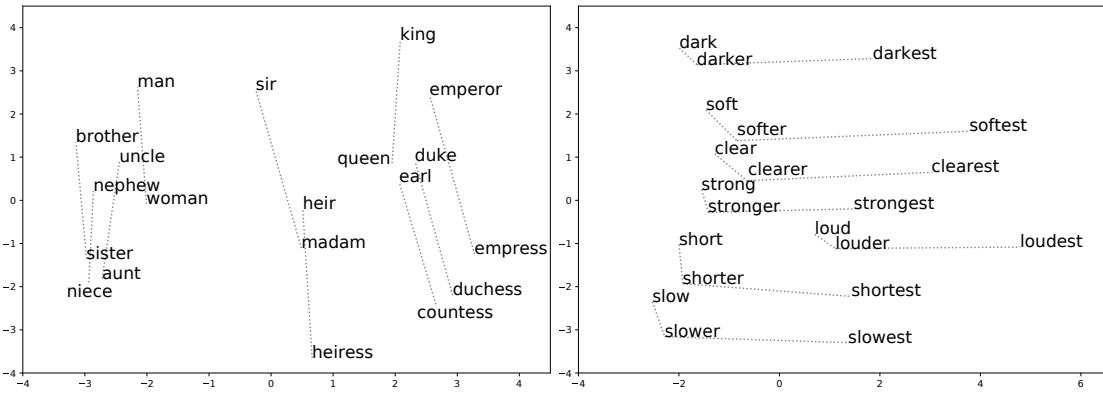
Table 14.1: Distributional statistics for *tezgüino* and five related terms

Figure 14.1: Lexical semantic relationships have regular linear structures in two dimensional projections of distributional statistics (Pennington et al., 2014).

7399 What other words fit into these contexts? How about: *loud*, *motor oil*, *tortillas*, *choices*,
 7400 *wine*? Each row of Table 14.1 is a vector that summarizes the contextual properties for
 7401 each word, with a value of one for contexts in which the word can appear, and a value of
 7402 zero for contexts in which it cannot. Based on these vectors, we can conclude: *wine* is very
 7403 similar to *tezgüino*; *motor oil* and *tortillas* are fairly similar to *tezgüino*; *loud* is completely
 7404 different.

7405 These vectors, which we will call **word representations**, describe the **distributional**
 7406 properties of each word. Does vector similarity imply semantic similarity? This is the **dis-**
 7407 **distributional hypothesis**, stated by Firth (1957) as: “You shall know a word by the company
 7408 it keeps.” The distributional hypothesis has stood the test of time: distributional statistics
 7409 are a core part of language technology today, because they make it possible to leverage
 7410 large amounts of unlabeled data to learn about rare words that do not appear in labeled
 7411 training data.

7412 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.

14.2 Design decisions for word representations

There are many approaches for computing word representations, but most can be distinguished on three main dimensions: the nature of the representation, the source of contextual information, and the estimation procedure.

14.2.1 Representation

Today, the dominant word representations are k -dimensional vectors of real numbers, known as **word embeddings**. (The name is due to the fact that each discrete word is embedded in a continuous vector space.) This representation dates back at least to the late 1980s (Deerwester et al., 1990), and is used in popular techniques such as WORD2VEC (Mikolov et al., 2013).

Word embeddings are well suited for neural networks, where they can be plugged in as inputs. They can also be applied in linear classifiers and structure prediction models (Turian et al., 2010), although it can be difficult to learn linear models that employ real-valued features (Kummerfeld et al., 2015). A popular alternative is bit-string representations, such as **Brown clusters** (§ 14.4), in which each word is represented by a variable-length sequence of zeros and ones (Brown et al., 1992).

Another representational question is whether to estimate one embedding per surface form (e.g., *bank*), or to estimate distinct embeddings for each word sense or synset. Intuitively, if word representations are to capture the meaning of individual words, then words with multiple meanings should have multiple embeddings. This can be achieved by integrating unsupervised clustering with word embedding estimation (Huang and Yates, 2012; Li and Jurafsky, 2015). However, Arora et al. (2016) argue that it is unnecessary to model distinct word senses explicitly, because the embeddings for each surface form are a linear combination of the embeddings of the underlying senses.

The moment one learns English, complications set in (Alfau, 1999)

| | |
|------------------------------|---|
| Brown Clusters | $\{one\}$ |
| WORD2VEC, $h = 2$ | $\{moment, one, English, complications\}$ |
| Structured WORD2VEC, $h = 2$ | $\{(moment, -2), (one, -1), (English, +1), (complications, +2)\}$ |
| Dependency contexts, | $\{(one, NSUBJ), (English, DOBJ), (moment, ACL^{-1})\}$ |

Table 14.2: Contexts for the word *learns*, according to various word representations. For dependency context, $(one, NSUBJ)$ means that there is a relation of type NSUBJ (nominal subject) *to* the word *one*, and $(moment, ACL^{-1})$ means that there is a relation of type ACL (adjectival clause) *from* the word *moment*.

7448 14.2.2 Context

7449 The distributional hypothesis says that word meaning is related to the “contexts” in which
 7450 the word appears, but context can be defined in many ways. In the *tezgiino* example, con-
 7451 texts are entire sentences, but in practice there are far too many sentences. At the oppo-
 7452 site extreme, the context could be defined as the immediately preceding word; this is the
 7453 context considered in Brown clusters. WORD2VEC takes an intermediate approach, using
 7454 local neighborhoods of words (e.g., $h = 5$) as contexts (Mikolov et al., 2013). Contexts
 7455 can also be much larger: for example, in **latent semantic analysis**, each word’s context
 7456 vector includes an entry per document, with a value of one if the word appears in the
 7457 document (Deerwester et al., 1990); in **explicit semantic analysis**, these documents are
 7458 Wikipedia pages (Gabrilovich and Markovitch, 2007).

7459 In structured WORD2VEC, context words are labeled by their position with respect to
 7460 the target word w_m (e.g., two words before, one word after), which makes the result-
 7461 ing word representations more sensitive to syntactic differences (Ling et al., 2015). An-
 7462 other way to incorporate syntax is to perform parsing as a preprocessing step, and then
 7463 form context vectors from the dependency edges (Levy and Goldberg, 2014) or predicate-
 7464 argument relations (Lin, 1998). The resulting context vectors for several of these methods
 7465 are shown in Table 14.2.

7466 The choice of context has a profound effect on the resulting representations, which
 7467 can be viewed in terms of word similarity. Applying latent semantic analysis (§ 14.3) to
 7468 contexts of size $h = 2$ and $h = 30$ yields the following nearest-neighbors for the word
 7469 *dog*.¹

- 7470 • ($h = 2$): *cat, horse, fox, pet, rabbit, pig, animal, mongrel, sheep, pigeon*

¹The example is from lecture slides by Marco Baroni, Alessandro Lenci, and Stefan Evert, who applied latent semantic analysis to the British National Corpus. You can find an online demo here: <http://clic.cimec.unitn.it/infomap-query/>

- 7471 • ($h = 30$): *kennel, puppy, pet, bitch, terrier, rottweiler, canine, cat, to bark, Alsatian*

7472 Which word list is better? Each word in the $h = 2$ list is an animal, reflecting the fact that
 7473 locally, the word *dog* tends to appear in the same contexts as other animal types (e.g., *pet*
 7474 *the dog, feed the dog*). In the $h = 30$ list, nearly everything is dog-related, including specific
 7475 breeds such as *rottweiler* and *Alsatian*. The list also includes words that are not animals
 7476 (*kennel*), and in one case (*to bark*), is not a noun at all. The 2-word context window is more
 7477 sensitive to syntax, while the 30-word window is more sensitive to topic.

7478 **14.2.3 Estimation**

7479 Word embeddings are estimated by optimizing some objective: the likelihood of a set of
 7480 unlabeled data (or a closely related quantity), or the reconstruction of a matrix of context
 7481 counts, similar to Table 14.1.

7482 **Maximum likelihood estimation** Likelihood-based optimization is derived from the
 7483 objective $\log p(\mathbf{w}; \mathbf{U})$, where $\mathbf{U} \in \mathbb{R}^{K \times V}$ is matrix of word embeddings, and $\mathbf{w} =$
 7484 $\{w_m\}_{m=1}^M$ is a corpus, represented as a list of M tokens. Recurrent neural network lan-
 7485 guage models (§ 6.3) optimize this objective directly, backpropagating to the input word
 7486 embeddings through the recurrent structure. However, state-of-the-art word embeddings
 7487 employ huge corpora with hundreds of billions of tokens, and recurrent architectures are
 7488 difficult to scale to such data. As a result, likelihood-based word embeddings are usually
 7489 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,

$$\min_{\mathbf{u}, \mathbf{v}} \|\mathbf{C} - \tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})\|_F, \quad [14.1]$$

7490 where $\tilde{\mathbf{C}}(\mathbf{u}, \mathbf{v})$ is the approximate reconstruction resulting from the embeddings \mathbf{u} and
 7491 \mathbf{v} , and $\|\mathbf{X}\|_F$ indicates the Frobenius norm, $\sum_{i,j} x_{i,j}^2$. Rather than factoring the matrix of
 7492 word-context counts directly, it is often helpful to transform these counts using information-
 7493 theoretic metrics such as **pointwise mutual information** (PMI), described in the next sec-
 7494 tion.

7495 **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),

$$\min_{\mathbf{U} \in \mathbb{R}^{V \times K}, \mathbf{S} \in \mathbb{R}^{K \times K}, \mathbf{V} \in \mathbb{R}^{|\mathcal{C}| \times K}} \|\mathbf{C} - \mathbf{USV}^\top\|_F \quad [14.2]$$

$$\text{s.t. } \mathbf{U}^\top \mathbf{U} = \mathbb{I} \quad [14.3]$$

$$\mathbf{V}^\top \mathbf{V} = \mathbb{I} \quad [14.4]$$

$$\forall i \neq j, \mathbf{S}_{i,j} = 0, \quad [14.5]$$

7496 where V is the size of the vocabulary, $|\mathcal{C}|$ is the number of contexts, and K is size of the
 7497 resulting embeddings, which are set equal to the rows of the matrix \mathbf{U} . The matrix \mathbf{S} is
 7498 constrained to be diagonal (these diagonal elements are called the singular values), and
 7499 the columns of the product \mathbf{SV}^\top provide descriptions of the contexts. Each element $c_{i,j}$ is
 7500 then reconstructed as a **bilinear product**,

$$c_{i,j} \approx \sum_{k=1}^K u_{i,k} s_k v_{j,k}. \quad [14.6]$$

7501 The objective is to minimize the sum of squared approximation errors. The orthonormality
 7502 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
 7503 uncorrelated, so that each dimension conveys unique information. Efficient implemen-
 7504 tations of truncated singular value decomposition are available in numerical computing
 7505 packages such as SCIPY and MATLAB.²

Latent semantic analysis is most effective when the count matrix is transformed before
 the application of SVD. One such transformation is **pointwise mutual information** (PMI;
 Church and Hanks, 1990), which captures the degree of association between word i and
 context j ,

$$\text{PMI}(i, j) = \log \frac{p(i, j)}{p(i)p(j)} = \log \frac{p(i | j)p(j)}{p(i)p(j)} = \log \frac{p(i | j)}{p(i)} \quad [14.7]$$

$$= \log \text{count}(i, j) - \log \sum_{i'=1}^V \text{count}(i', j) \quad [14.8]$$

$$- \log \sum_{j' \in \mathcal{C}} \text{count}(i, j') + \log \sum_{i'=1}^V \sum_{j' \in \mathcal{C}} \text{count}(i', j'). \quad [14.9]$$

7506 The pointwise mutual information can be viewed as the logarithm of the ratio of the con-
 7507 ditional probability of word i in context j to the marginal probability of word i in all

²An important implementation detail is to represent \mathbf{C} as a **sparse matrix**, so that the storage cost is equal to the number of non-zero entries, rather than the size $V \times |\mathcal{C}|$.

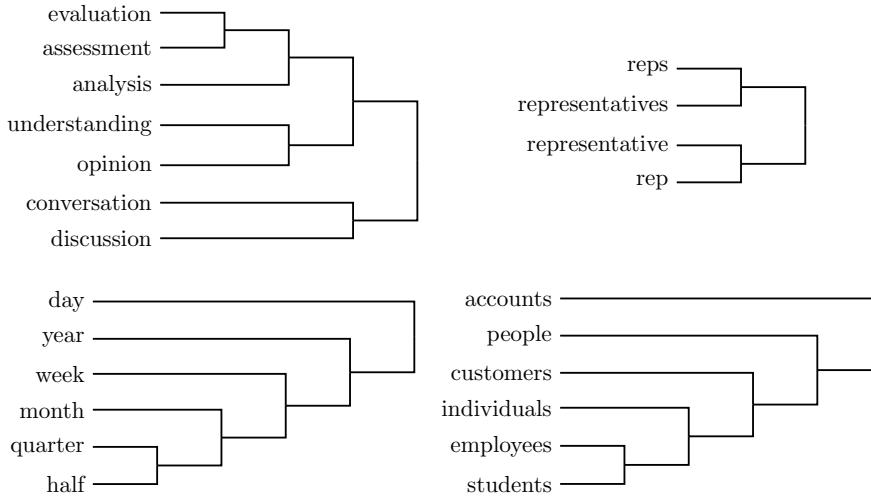


Figure 14.2: Subtrees produced by bottom-up Brown clustering on news text (Miller et al., 2004).

7508 contexts. When word i is statistically associated with context j , the ratio will be greater
 7509 than one, so $\text{PMI}(i, j) > 0$. The PMI transformation focuses latent semantic analysis on re-
 7510 constructing strong word-context associations, rather than on reconstructing large counts.

7511 The PMI is negative when a word and context occur together less often than if they
 7512 were independent, but such negative correlations are unreliable because counts of rare
 7513 events have high variance. Furthermore, the PMI is undefined when $\text{count}(i, j) = 0$. One
 7514 solution to these problems is to use the **Positive PMI** (PPMI),

$$\text{PPMI}(i, j) = \begin{cases} \text{PMI}(i, j), & p(i | j) > p(i) \\ 0, & \text{otherwise.} \end{cases} \quad [14.10]$$

7515 Bullinaria and Levy (2007) compare a range of matrix transformations for latent se-
 7516 mantic analysis, using a battery of tasks related to word meaning and word similarity
 7517 (for more on evaluation, see § 14.6). They find that PPMI-based latent semantic analysis
 7518 yields strong performance on a battery of tasks related to word meaning: for example,
 7519 PPMI-based LSA vectors can be used to solve multiple-choice word similarity questions
 7520 from the Test of English as a Foreign Language (TOEFL), obtaining 85% accuracy.

7521 14.4 Brown clusters

7522 Learning algorithms like perceptron and conditional random fields often perform better
 7523 with discrete feature vectors. A simple way to obtain discrete representations from distri-

| bitstring | ten most frequent words |
|----------------------|--|
| 01111010 0111 | <i>excited thankful grateful stoked pumped anxious hyped psyched exited geeked</i> |
| 01111010 100 | <i>talking talkin complaining talkn bitching tlkn tlkin bragging raving +k</i> |
| 01111010 1010 | <i>thinking thinkin dreaming worrying thinkn speakin reminiscing dreamin daydreaming fantasizing</i> |
| 01111010 1011 | <i>saying sayin suggesting stating sayn jokin talmbout implying insisting 5'2</i> |
| 01111010 1100 | <i>wonder dunno wondered duno donno dno dono wonda wounder dunnoe</i> |
| 01111010 1101 | <i>wondering wonders debating deciding pondering unsure wonderin debatin woundering wondern</i> |
| 01111010 1110 | <i>sure suree suuure suure sure- surre sures shuree</i> |

Table 14.3: Fragment of a Brown clustering of Twitter data (Owoputi et al., 2013). Each row is a leaf in the tree, showing the ten most frequent words. This part of the tree emphasizes verbs of communicating and knowing, especially in the present participle. Each leaf node includes orthographic variants (*thinking*, *thinkin*, *thinkn*), semantically related terms (*excited*, *thankful*, *grateful*), and some outliers (*5'2*, *+k*). See http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html for more.

7524 butional statistics is by clustering (§ 5.1.1), so that words in the same cluster have similar
 7525 distributional statistics. This can help in downstream tasks, by sharing features between
 7526 all words in the same cluster. However, there is an obvious tradeoff: if the number of clus-
 7527 ters is too small, the words in each cluster will not have much in common; if the number
 7528 of clusters is too large, then the learner will not see enough examples from each cluster to
 7529 generalize.

7530 A solution to this problem is **hierarchical clustering**: using the distributional statistics
 7531 to induce a tree-structured representation. Fragments of **Brown cluster** trees are shown in
 7532 Figure 14.2 and Table 14.3. Each word’s representation consists of a binary string describ-
 7533 ing a path through the tree: 0 for taking the left branch, and 1 for taking the right branch.
 7534 In the subtree in the upper right of the figure, the representation of the word *conversation*
 7535 is 10; the representation of the word *assessment* is 0001. Bitstring prefixes capture simila-
 7536 rity at varying levels of specificity, and it is common to use the first eight, twelve, sixteen,
 7537 and twenty bits as features in tasks such as named entity recognition (Miller et al., 2004)
 7538 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 :

$$\log p(\mathbf{w}; \mathbf{k}) \approx \sum_{m=1}^M \log p(w_m | w_{m-1}; \mathbf{k}) \quad [14.11]$$

$$\triangleq \sum_{m=1}^M \log p(w_m | k_{w_m}) + \log p(k_{w_m} | k_{w_{m-1}}). \quad [14.12]$$

7539 This is similar to a hidden Markov model, with the crucial difference that each word can
 7540 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**,

$$I(\mathbf{k}) = \sum_{k_1=1}^K \sum_{k_2=1}^K p(k_1, k_2) \times \text{PMI}(k_1, k_2) \quad [14.13]$$

$$p(k_1, k_2) = \frac{\text{count}(k_1, k_2)}{\sum_{k_1'=1}^K \sum_{k_2'=1}^K \text{count}(k_1', k_2')},$$

7541 where $p(k_1, k_2)$ is the joint probability of a bigram involving a word in cluster k_1 followed
 7542 by a word in k_2 . This probability and the PMI are both computed from the co-occurrence
 7543 counts between clusters. After each merger, the co-occurrence vectors for the merged
 7544 clusters are simply added up, so that the next optimal merger can be found efficiently.

7545 This bottom-up procedure requires iterating over the entire vocabulary, and evaluating
 7546 K_t^2 possible mergers at each step, where K_t is the current number of clusters at step t
 7547 of the algorithm. Furthermore, computing the score for each merger involves a sum over
 7548 K_t^2 clusters. The maximum number of clusters is $K_0 = V$, which occurs when every word
 7549 is in its own cluster at the beginning of the algorithm. The time complexity is thus $\mathcal{O}(V^5)$.

7550 To avoid this complexity, practical implementations use a heuristic approximation
 7551 called **exchange clustering**. The K most common words are placed in clusters of their
 7552 own at the beginning of the process. We then consider the next most common word, and
 7553 merge it with one of the existing clusters. This continues until the entire vocabulary has
 7554 been incorporated, at which point the K clusters are merged down to a single cluster,
 7555 forming a tree. The algorithm never considers more than $K + 1$ clusters at any step, and
 7556 the complexity is $\mathcal{O}(VK + V \log V)$, with the second term representing the cost of sorting

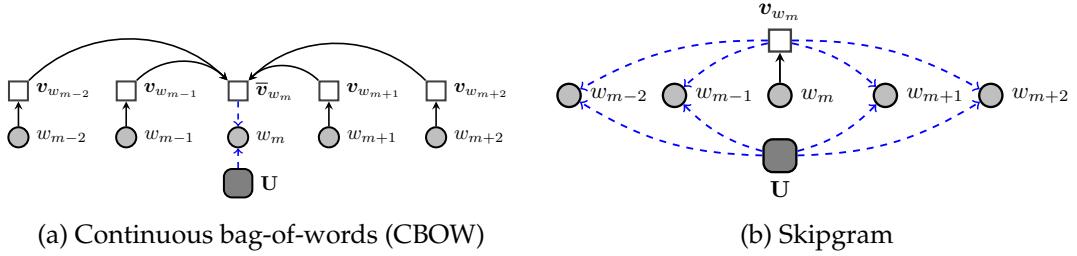


Figure 14.3: The CBOW and skipgram variants of WORD2VEC. The parameter \mathbf{U} is the matrix of word embeddings, and each v_m is the context embedding for word w_m .

7557 the words at the beginning of the algorithm. For more details on the algorithm, see Liang
 7558 (2005).

7559 14.5 Neural word embeddings

7560 Neural word embeddings combine aspects of the previous two methods: like latent se-
 7561 mantic analysis, they are a continuous vector representation; like Brown clusters, they are
 7562 trained from a likelihood-based objective. Let the vector u_i represent the K -dimensional
 7563 **embedding** for word i , and let v_j represent the K -dimensional embedding for context
 7564 j . The inner product $u_i \cdot v_j$ represents the compatibility between word i and context j .
 7565 By incorporating this inner product into an approximation to the log-likelihood of a cor-
 7566 pus, it is possible to estimate both parameters by backpropagation. WORD2VEC (Mikolov
 7567 et al., 2013) includes two such approximations: continuous bag-of-words (CBOW) and
 7568 skipgrams.

7569 14.5.1 Continuous bag-of-words (CBOW)

7570 In recurrent neural network language models, each word w_m is conditioned on a recurrently-
 7571 updated state vector, which is based on word representations going all the way back to the
 7572 beginning of the text. The **continuous bag-of-words (CBOW)** model is a simplification:
 7573 the local context is computed as an average of embeddings for words in the immediate
 7574 neighborhood $m - h, m - h + 1, \dots, m + h - 1, m + h$,

$$\bar{v}_m = \frac{1}{2h} \sum_{n=1}^h v_{w_{m+n}} + v_{w_{m-n}}. \quad [14.14]$$

7575 Thus, CBOW is a bag-of-words model, because the order of the context words does not
 7576 matter; it is continuous, because rather than conditioning on the words themselves, we
 7577 condition on a continuous vector constructed from the word embeddings. The parameter
 7578 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,

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \log p(w_m | w_{m-h}, w_{m-h+1}, \dots, w_{m+h-1}, w_{m+h}) \quad [14.15]$$

$$= \sum_{m=1}^M \log \frac{\exp(\mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m)}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m)} \quad [14.16]$$

$$= \sum_{m=1}^M \mathbf{u}_{w_m} \cdot \bar{\mathbf{v}}_m - \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \bar{\mathbf{v}}_m). \quad [14.17]$$

7579 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:

$$\log p(\mathbf{w}) \approx \sum_{m=1}^M \sum_{n=1}^{h_m} \log p(w_{m-n} | w_m) + \log p(w_{m+n} | w_m) \quad [14.18]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \log \frac{\exp(\mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} + \log \frac{\exp(\mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m})}{\sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m})} \quad [14.19]$$

$$= \sum_{m=1}^M \sum_{n=1}^{h_m} \mathbf{u}_{w_{m-n}} \cdot \mathbf{v}_{w_m} + \mathbf{u}_{w_{m+n}} \cdot \mathbf{v}_{w_m} - 2 \log \sum_{j=1}^V \exp(\mathbf{u}_j \cdot \mathbf{v}_{w_m}). \quad [14.20]$$

7580 In the skipgram approximation, each word is generated multiple times; each time it is con-
 7581 ditioned only on a single word. This makes it possible to avoid averaging the word vec-
 7582 tors, as in the CBOW model. The local neighborhood size h_m is randomly sampled from
 7583 a uniform categorical distribution over the range $\{1, 2, \dots, h_{\max}\}$; Mikolov et al. (2013) set
 7584 $h_{\max} = 10$. Because the neighborhood grows outward with h , this approach has the effect
 7585 of weighting near neighbors more than distant ones. Skipgram performs better on most
 7586 evaluations than CBOW (see § 14.6 for details of how to evaluate word representations),
 7587 but CBOW is faster to train (Mikolov et al., 2013).

7588 14.5.3 Computational complexity

7589 The WORD2VEC models can be viewed as an efficient alternative to recurrent neural net-
 7590 work language models, which involve a recurrent state update whose time complexity
 7591 is quadratic in the size of the recurrent state vector. CBOW and skipgram avoid this
 7592 computation, and incur only a linear time complexity in the size of the word and con-
 7593 text representations. However, all three models compute a normalized probability over
 7594 word tokens; a naïve implementation of this probability requires summing over the entire

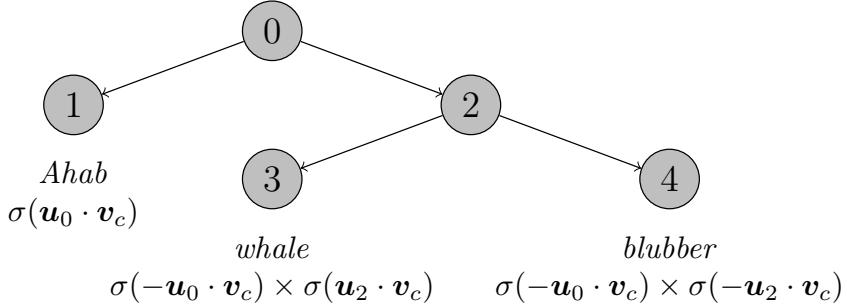


Figure 14.4: A fragment of a hierarchical softmax tree. The probability of each word is computed as a product of probabilities of local branching decisions in the tree.

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.

7600 Hierarchical softmax

In Brown clustering, the vocabulary is organized into a binary tree. Mnih and Hinton (2008) show that the normalized probability over words in the vocabulary can be reparametrized as a probability over paths through such a tree. This hierarchical softmax probability is computed as a product of binary decisions over whether to move left or right through the tree, with each binary decision represented as a sigmoid function of the inner product between the context embedding \mathbf{v}_c and an output embedding associated with the node \mathbf{u}_n ,

$$\Pr(\text{left at } n \mid c) = \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) \quad [14.21]$$

$$\Pr(\text{right at } n \mid c) = 1 - \sigma(\mathbf{u}_n \cdot \mathbf{v}_c) = \sigma(-\mathbf{u}_n \cdot \mathbf{v}_c), \quad [14.22]$$

where σ refers to the sigmoid function, $\sigma(x) = \frac{1}{1+\exp(-x)}$. The range of the sigmoid is the interval $(0, 1)$, and $1 - \sigma(x) = \sigma(-x)$.

As shown in Figure 14.4, the probability of generating each word is redefined as the product of the probabilities across its path. The sum of all such path probabilities is guaranteed to be one, for any context vector $\mathbf{v}_c \in \mathbb{R}^K$. In a balanced binary tree, the depth is logarithmic in the number of leaf nodes, and thus the number of multiplications is equal to $\mathcal{O}(\log V)$. The number of non-leaf nodes is equal to $\mathcal{O}(2V - 1)$, so the number of parameters to be estimated increases by only a small multiple. The tree can be constructed using an incremental clustering procedure similar to hierarchical Brown clusters (Mnih

7610 and Hinton, 2008), or by using the Huffman (1952) encoding algorithm for lossless com-
 7611 pression.

7612 **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**:

$$\psi(i, j) = \log \sigma(\mathbf{u}_i \cdot \mathbf{v}_j) + \sum_{i' \in \mathcal{W}_{\text{neg}}} \log(1 - \sigma(\mathbf{u}_{i'} \cdot \mathbf{v}_j)), \quad [14.23]$$

7613 where $\psi(i, j)$ is the score for word i in context j , and \mathcal{W}_{neg} is the set of negative samples.
 7614 The objective is to maximize the sum over the corpus, $\sum_{m=1}^M \psi(w_m, c_m)$, where w_m is
 7615 token m and c_m is the associated context.

7616 The set of negative samples \mathcal{W}_{neg} is obtained by sampling from a unigram language
 7617 model. Mikolov et al. (2013) construct this unigram language model by exponentiating
 7618 the empirical word probabilities, setting $\hat{p}(i) \propto (\text{count}(i))^{\frac{3}{4}}$. This has the effect of redis-
 7619 tributing probability mass from common to rare words. The number of negative samples
 7620 increases the time complexity of training by a constant factor. Mikolov et al. (2013) report
 7621 that 5-20 negative samples works for small training sets, and that two to five samples
 7622 suffice for larger corpora.

7623 **14.5.4 Word embeddings as matrix factorization**

7624 The negative sampling objective in Equation 14.23 can be justified as an efficient approx-
 7625 imation to the log-likelihood, but it is also closely linked to the matrix factorization ob-
 7626 jective employed in latent semantic analysis. For a matrix of word-context pairs in which
 7627 all counts are non-zero, negative sampling is equivalent to factorization of the matrix M ,
 7628 where $M_{ij} = \text{PMI}(i, j) - \log k$: each cell in the matrix is equal to the pointwise mutual
 7629 information of the word and context, shifted by $\log k$, with k equal to the number of neg-
 7630 ative samples (Levy and Goldberg, 2014). For word-context pairs that are not observed in
 7631 the data, the pointwise mutual information is $-\infty$, but this can be addressed by consid-
 7632 ering only PMI values that are greater than $\log k$, resulting in a matrix of **shifted positive**
 7633 **pointwise mutual information**,

$$M_{ij} = \max(0, \text{PMI}(i, j) - \log k). \quad [14.24]$$

7634 Word embeddings are obtained by factoring this matrix with truncated singular value
 7635 decomposition.

| word 1 | word 2 | similarity |
|--------------------|----------------|------------|
| <i>love</i> | <i>sex</i> | 6.77 |
| <i>stock</i> | <i>jaguar</i> | 0.92 |
| <i>money</i> | <i>cash</i> | 9.15 |
| <i>development</i> | <i>issue</i> | 3.97 |
| <i>lad</i> | <i>brother</i> | 4.46 |

Table 14.4: Subset of the WS-353 (Finkelstein et al., 2002) dataset of word similarity ratings (examples from Faruqui et al. (2016)).

GloVe (“global vectors”) are a closely related approach (Pennington et al., 2014), in which the matrix to be factored is constructed from log co-occurrence counts, $M_{ij} = \log \text{count}(i, j)$. The word embeddings are estimated by minimizing the sum of squares,

$$\begin{aligned} \min_{\mathbf{u}, \mathbf{v}, b, \tilde{b}} \quad & \sum_{j=1}^V \sum_{j \in \mathcal{C}} f(M_{ij}) \left(\widehat{\log M_{ij}} - \log M_{ij} \right)^2 \\ \text{s.t.} \quad & \widehat{\log M_{ij}} = \mathbf{u}_i \cdot \mathbf{v}_j + b_i + \tilde{b}_j, \end{aligned} \quad [14.25]$$

where b_i and \tilde{b}_j are offsets for word i and context j , which are estimated jointly with the embeddings \mathbf{u} and \mathbf{v} . The weighting function $f(M_{ij})$ is set to be zero at $M_{ij} = 0$, thus avoiding the problem of taking the logarithm of zero counts; it saturates at $M_{ij} = m_{\max}$, thus avoiding the problem of overcounting common word-context pairs. This heuristic turns out to be critical to the method’s performance.

The time complexity of sparse matrix reconstruction is determined by the number of non-zero word-context counts. Pennington et al. (2014) show that this number grows sublinearly with the size of the dataset: roughly $\mathcal{O}(N^{0.8})$ for typical English corpora. In contrast, the time complexity of WORD2VEC is linear in the corpus size. Computing the co-occurrence counts also requires linear time in the size of the corpus, but this operation can easily be parallelized using MapReduce-style algorithms (Dean and Ghemawat, 2008).

14.6 Evaluating word embeddings

Distributed word representations can be evaluated in two main ways. **Intrinsic** evaluations test whether the representations cohere with our intuitions about word meaning. **Extrinsic** evaluations test whether they are useful for downstream tasks, such as sequence labeling.

7652 **14.6.1 Intrinsic evaluations**

7653 A basic question for word embeddings is whether the similarity of words i and j is re-
 7654 flected in the similarity of the vectors \mathbf{u}_i and \mathbf{u}_j . **Cosine similarity** is typically used to
 7655 compare two word embeddings,

$$\cos(\mathbf{u}_i, \mathbf{u}_j) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|_2 \times \|\mathbf{u}_j\|_2}. \quad [14.26]$$

7656 For any embedding method, we can evaluate whether the cosine similarity of word em-
 7657 beddings is correlated with human judgments of word similarity. The WS-353 dataset (Finkel-
 7658 stein et al., 2002) includes similarity scores for 353 word pairs (Table 14.4). To test the
 7659 accuracy of embeddings for rare and morphologically complex words, Luong et al. (2013)
 7660 introduce a dataset of “rare words.” Outside of English, word similarity resources are lim-
 7661 ited, mainly consisting of translations of WS-353 and the related SimLex-999 dataset (Hill
 7662 et al., 2015).

7663 Word analogies (e.g., *king:queen :: man:woman*) have also been used to evaluate word
 7664 embeddings (Mikolov et al., 2013). In this evaluation, the system is provided with the first
 7665 three parts of the analogy ($i_1 : j_1 :: i_2 : ?$), and the final element is predicted by finding the
 7666 word embedding most similar to $\mathbf{u}_{i_1} - \mathbf{u}_{j_1} + \mathbf{u}_{i_2}$. Another evaluation tests whether word
 7667 embeddings are related to broad lexical semantic categories called **supersenses** (Ciaramita
 7668 and Johnson, 2003): verbs of motion, nouns that describe animals, nouns that describe
 7669 body parts, and so on. These supersenses are annotated for English synsets in Word-
 7670 Net (Fellbaum, 2010). This evaluation is implemented in the QVEC metric, which tests
 7671 whether the matrix of supersenses can be reconstructed from the matrix of word embed-
 7672 dings (Tsvetkov et al., 2015).

7673 Levy et al. (2015) compared several dense word representations for English — includ-
 7674 ing latent semantic analysis, WORD2VEC, and GloVe — using six word similarity metrics
 7675 and two analogy tasks. None of the embeddings outperformed the others on every task,
 7676 but skipgrams were the most broadly competitive. Hyperparameter tuning played a key
 7677 role: any method will perform badly if the wrong hyperparameters are used. Relevant
 7678 hyperparameters include the embedding size, as well as algorithm-specific details such
 7679 as the neighborhood size and the number of negative samples.

7680 **14.6.2 Extrinsic evaluations**

7681 Word representations contribute to downstream tasks like sequence labeling and docu-
 7682 ment classification by enabling generalization across words. The use of distributed repre-
 7683 sentations as features is a form of **semi-supervised learning**, in which performance on a
 7684 supervised learning problem is augmented by learning distributed representations from
 7685 unlabeled data (Miller et al., 2004; Koo et al., 2008; Turian et al., 2010). These **pre-trained**
 7686 **word representations** can be used as features in a linear prediction model, or as the input

layer in a neural network, such as a Bi-LSTM tagging model (§ 7.6). Word representations can be evaluated by the performance of the downstream systems that consume them: for example, GloVe embeddings are convincingly better than Latent Semantic Analysis as features in the downstream task of named entity recognition (Pennington et al., 2014). Unfortunately, extrinsic and intrinsic evaluations do not always point in the same direction, and the best word representations for one downstream task may perform poorly on another task (Schnabel et al., 2015).

When word representations are updated from labeled data in the downstream task, they are said to be **fine-tuned**. When labeled data is plentiful, pre-training may be unnecessary; when labeled data is scarce, fine-tuning may lead to overfitting. Various combinations of pre-training and fine-tuning can be employed. Pre-trained embeddings can be used as initialization before fine-tuning, and this can substantially improve performance (Lample et al., 2016). Alternatively, both fine-tuned and pre-trained embeddings can be used as inputs in a single model (Kim, 2014).

In semi-supervised scenarios, pretrained word embeddings can be replaced by “contextualized” word representations (Peters et al., 2018). These contextualized representations are set to the hidden states of a deep bi-directional LSTM, which is trained as a bi-directional language model, motivating the name **ELMo (embeddings from language models)**. By running the language model, we obtain contextualized word representations, which can then be used as the base layer in a supervised neural network for any task. This approach yields significant gains over pretrained word embeddings on several tasks, presumably because the contextualized embeddings use unlabeled data to learn how to integrate linguistic context into the base layer of the supervised neural network.

14.6.3 Fairness and bias

Figure 14.1 shows how word embeddings can capture analogies such as *man:woman :: king:queen*. While *king* and *queen* are gender-specific by definition, other professions or titles are associated with genders and other groups merely by statistical tendency. This statistical tendency may be a fact about the world (e.g., professional baseball players are usually men), or a fact about the text corpus (e.g., there are professional basketball leagues for both women and men, but the men’s basketball is written about far more often).

There is now considerable evidence that word embeddings do indeed encode such biases. Bolukbasi et al. (2016) show that the words most aligned with the vector difference *she – he* are stereotypically female professions *homemaker, nurse, receptionist*; in the other direction are *maestro, skipper, protege*. Caliskan et al. (2017) systematize this observation by showing that biases in word embeddings align with well-validated gender stereotypes. Garg et al. (2018) extend these results to ethnic stereotypes of Asian Americans, and provide a historical perspective on how stereotypes evolve over 100 years of text data.

Because word embeddings are the input layer for many other natural language pro-

cessing systems, these findings highlight the risk that natural language processing will replicate and amplify biases in the world, as well as in text. If, for example, word embeddings encode the belief that women are as unlikely to be computer programmers as they are to be nephews, then software is unlikely to successfully parse, translate, index, and extract those cases in which women do indeed program computers. For example, in such cases, contemporary NLP systems often fail to properly resolve pronoun references (Rudinger et al., 2018; Zhao et al., 2018). (The task of pronoun resolution is described in depth in chapter 15.) Such biases can have profound consequences: for example, search engines are more likely to yield personalized advertisements for public arrest records when queried with names that are statistically associated with African Americans (Sweeney, 2013). There is now an active research literature on “debiasing” machine learning and natural language processing, as evidenced by the growth of annual meetings such as Fairness, Accountability, and Transparency in Machine Learning (FAT/ML). However, given that the ultimate source of these biases is the text itself, it may be too much to hope for a purely algorithmic solution. There is no substitute for critical thought about the inputs to natural language processing systems – and the uses of their outputs.

14.7 Distributed representations beyond distributional statistics

Distributional word representations can be estimated from huge unlabeled datasets, thereby covering many words that do not appear in labeled data: for example, GloVe embeddings are estimated from 800 billion tokens of web data,³ while the largest labeled datasets for NLP tasks are on the order of millions of tokens. Nonetheless, even a dataset of hundreds of billions of tokens will not cover every word that may be encountered in the future. Furthermore, many words will appear only a few times, making their embeddings unreliable. Many languages exceed English in morphological complexity, and thus have lower token-to-type ratios. When this problem is coupled with small training corpora, it becomes especially important to leverage other sources of information beyond distributional statistics.

14.7.1 Word-internal structure

One solution is to incorporate word-internal structure into word embeddings. Purely distributional approaches consider words as atomic units, but in fact, many words have internal structure, so that their meaning can be **composed** from the representations of sub-word units. Consider the following terms, all of which are missing from Google’s pre-trained WORD2VEC embeddings:⁴

³<http://commoncrawl.org/>

⁴<https://code.google.com/archive/p/word2vec/>, accessed September 20, 2017

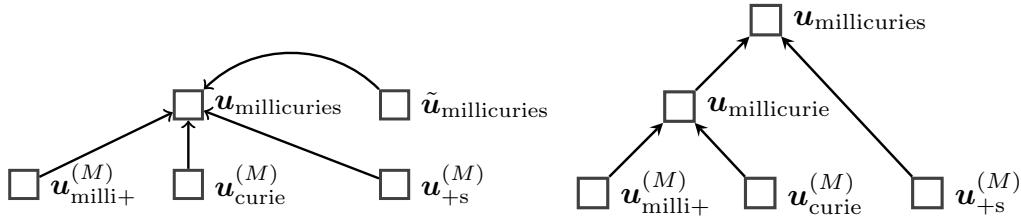


Figure 14.5: Two architectures for building word embeddings from subword units. On the left, morpheme embeddings $u^{(m)}$ are combined by addition with the non-compositional word embedding \tilde{u} (Botha and Blunsom, 2014). On the right, morpheme embeddings are combined in a recursive neural network (Luong et al., 2013).

7758 ***millicuries*** This word has **morphological** structure (see § 9.1.2 for more on morphology):
 7759 the prefix *milli-* indicates an amount, and the suffix *-s* indicates a plural. (A *millicurie*
 7760 is an unit of radioactivity.)

7761 ***caesium*** This word is a single morpheme, but the characters *-ium* are often associated
 7762 with chemical elements. (*Caesium* is the British spelling of a chemical element,
 7763 spelled *cesium* in American English.)

7764 ***IAEA*** This term is an acronym, as suggested by the use of capitalization. The prefix *I-* fre-
 7765 quently refers to international organizations, and the suffix *-A* often refers to agen-
 7766 cies or associations. (*IAEA* is the International Atomic Energy Agency.)

7767 ***Zhezhan*** This term is in title case, suggesting the name of a person or place, and the
 7768 character bigram *zh* indicates that it is likely a transliteration. (*Zhezhan* is a mining
 7769 facility in Kazakhstan.)

7770 How can word-internal structure be incorporated into word representations? One
 7771 approach is to construct word representations from embeddings of the characters or mor-
 7772 phemes. For example, if word i has morphological segments \mathcal{M}_i , then its embedding can
 7773 be constructed by addition (Botha and Blunsom, 2014),

$$\mathbf{u}_i = \tilde{\mathbf{u}}_i + \sum_{j \in \mathcal{M}_i} \mathbf{u}_j^{(M)}, \quad [14.27]$$

7774 where $\mathbf{u}_m^{(M)}$ is a morpheme embedding and $\tilde{\mathbf{u}}_i$ is a non-compositional embedding of the
 7775 whole word, which is an additional free parameter of the model (Figure 14.5, left side).
 7776 All embeddings are estimated from a **log-bilinear language model** (Mnih and Hinton,
 7777 2007), which is similar to the CBOW model (§ 14.5), but includes only contextual informa-
 7778 tion from preceding words. The morphological segments are obtained using an unsuper-
 7779 vised segmenter (Creutz and Lagus, 2007). For words that do not appear in the training

7780 data, the embedding can be constructed directly from the morphemes, assuming that each
 7781 morpheme appears in some other word in the training data. The free parameter \tilde{u} adds
 7782 flexibility: words with similar morphemes are encouraged to have similar embeddings,
 7783 but this parameter makes it possible for them to be different.

7784 Word-internal structure can be incorporated into word representations in various other
 7785 ways. Here are some of the main parameters.

7786 **Subword units.** Examples like *IAEA* and *Zhezhgan* are not based on morphological com-
 7787 position, and a morphological segmenter is unlikely to identify meaningful sub-
 7788 word units for these terms. Rather than using morphemes for subword embeddings,
 7789 one can use characters (Santos and Zadrozny, 2014; Ling et al., 2015; Kim et al., 2016),
 7790 character n -grams (Wieting et al., 2016; Bojanowski et al., 2017), and **byte-pair en-**
 7791 **codings**, a compression technique which captures frequent substrings (Gage, 1994;
 7792 Sennrich et al., 2016).

7793 **Composition.** Combining the subword embeddings by addition does not differentiate
 7794 between orderings, nor does it identify any particular morpheme as the root. A
 7795 range of more flexible compositional models have been considered, including re-
 7796 currence (Ling et al., 2015), convolution (Santos and Zadrozny, 2014; Kim et al.,
 7797 2016), and **recursive neural networks** (Luong et al., 2013), in which representa-
 7798 tions of progressively larger units are constructed over a morphological parse, e.g.
 7799 $((\text{milli}+\text{curie})+\text{s})$, $((\text{in}+\text{flam})+\text{able})$, $(\text{in}+(\text{vis}+\text{ible}))$. A recursive embedding model is
 7800 shown in the right panel of Figure 14.5.

7801 **Estimation.** Estimating subword embeddings from a full dataset is computationally ex-
 7802 pensive. An alternative approach is to train a subword model to match pre-trained
 7803 word embeddings (Cotterell et al., 2016; Pinter et al., 2017). To train such a model, it
 7804 is only necessary to iterate over the vocabulary, and the not the corpus.

7805 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,

$$\min_{\mathbf{U}} \sum_{j=1}^V \|\mathbf{u}_i - \hat{\mathbf{u}}_i\|_2 + \sum_{(i,j) \in \mathcal{L}} \beta_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|_2, \quad [14.28]$$

7806 where \hat{u}_i is the pretrained embedding of word i , and $\mathcal{L} = \{(i, j)\}$ is a lexicon of word
 7807 relations. The hyperparameter β_{ij} controls the importance of adjacent words having
 7808 similar embeddings; Faruqui et al. (2015) set it to the inverse of the degree of word i ,
 7809 $\beta_{ij} = |\{j : (i, j) \in \mathcal{L}\}|^{-1}$. Retrofitting improves performance on a range of intrinsic evalua-
 7810 tions, and gives small improvements on an extrinsic document classification task.

7811 14.8 Distributed representations of multiword units

7812 Can distributed representations extend to phrases, sentences, paragraphs, and beyond?
 7813 Before exploring this possibility, recall the distinction between distributed and distri-
 7814 butional representations. Neural embeddings such as WORD2VEC are both distributed
 7815 (vector-based) and distributional (derived from counts of words in context). As we con-
 7816 sider larger units of text, the counts decrease: in the limit, a multi-paragraph span of text
 7817 would never appear twice, except by plagiarism. Thus, the meaning of a large span of
 7818 text cannot be determined from distributional statistics alone; it must be computed com-
 7819 positionally from smaller spans. But these considerations are orthogonal to the question
 7820 of whether distributed representations — dense numerical vectors — are sufficiently ex-
 7821 pressive to capture the meaning of phrases, sentences, and paragraphs.

7822 14.8.1 Purely distributional methods

7823 Some multiword phrases are non-compositional: the meaning of such phrases is not de-
 7824 rived from the meaning of the individual words using typical compositional semantics.
 7825 This includes proper nouns like *San Francisco* as well as idiomatic expressions like *kick*
 7826 *the bucket* (Baldwin and Kim, 2010). For these cases, purely distributional approaches
 7827 can work. A simple approach is to identify multiword units that appear together fre-
 7828 quently, and then treat these units as words, learning embeddings using a technique such
 7829 as WORD2VEC.

7830 The problem of identifying multiword units is sometimes called **collocation extrac-**
 7831 **tion.** A good collocation has high **pointwise mutual information** (PMI), $\log p(w_t =$
 7832 $i | w_{t-1} = j) - \log p(w_t = i)$. For example, *Naïve Bayes* is a good collocation because
 7833 $p(w_t = Bayes | w_{t-1} = naïve)$ is much larger than $p(w_t = Bayes)$. Multiword collocation
 7834 can be performed by greedily extracting and grouping the collocations with the maxi-
 7835 mum PMI: for example, *mutual information* might first be extracted as a collocation and
 7836 grouped into a single word type *mutual_information*; then *pointwise mutual_information* can
 7837 be extracted later. After identifying such units, they can be treated as words when esti-
 7838 mating skipgram embeddings. Mikolov et al. (2013) show that the resulting embeddings
 7839 perform reasonably on a task of solving phrasal analogies, e.g. *New York : New York Times*
 7840 :: *Baltimore : Baltimore Sun*.

this was the only way
 it was the only way
 it was her turn to blink
 it was hard to tell
 it was time to move on
 he had to do it again
 they all looked at each other
 they all turned to look back
 they both turned to face him
they both turned and walked away

Figure 14.6: By interpolating between the distributed representations of two sentences (in bold), it is possible to generate grammatical sentences that combine aspects of both (Bowman et al., 2016)

7841 14.8.2 Distributional-compositional hybrids

7842 To move beyond short multiword phrases, composition is necessary. A simple but sur-
 7843 prisingly powerful approach is to represent a sentence with the average of its word em-
 7844 beddings (Mitchell and Lapata, 2010). This can be considered a hybrid of the distribu-
 7845 tional and compositional approaches to semantics: the word embeddings are computed
 7846 distributionally, and then the sentence representation is computed by composition.

7847 The WORD2VEC approach can be stretched considerably further, embedding entire
 7848 sentences using a model similar to skipgrams, in the “skip-thought” model of Kiros et al.
 7849 (2015). Each sentence is *encoded* into a vector using a recurrent neural network: the encod-
 7850 ing of sentence t is set to the RNN hidden state at its final token, $h_{M_t}^{(t)}$. This vector is then
 7851 a parameter in a *decoder* model that is used to generate the previous and subsequent sen-
 7852 tences: the decoder is another recurrent neural network, which takes the encoding of the
 7853 neighboring sentence as an additional parameter in its recurrent update. (This **encoder-**
 7854 **decoder model** is discussed at length in chapter 18.) The encoder and decoder are trained
 7855 simultaneously from a likelihood-based objective, and the trained encoder can be used to
 7856 compute a distributed representation of any sentence. Skip-thought can also be viewed
 7857 as a hybrid of distributional and compositional approaches: the vector representation of
 7858 each sentence is computed compositionally from the representations of the individual
 7859 words, but the training objective is distributional, based on sentence co-occurrence across
 7860 a corpus.

7861 **Autoencoders** are a variant of encoder-decoder models in which the decoder is trained
 7862 to produce the same text that was originally encoded, using only the distributed encod-
 7863 ing vector (Li et al., 2015). The encoding acts as a bottleneck, so that generalization is
 7864 necessary if the model is to successfully fit the training data. In **denoising autoencoders**,

7865 the input is a corrupted version of the original sentence, and the auto-encoder must re-
 7866 construct the uncorrupted original (Vincent et al., 2010; Hill et al., 2016). By interpolating
 7867 between distributed representations of two sentences, $\alpha \mathbf{u}_i + (1 - \alpha) \mathbf{u}_j$, it is possible to gen-
 7868 erate sentences that combine aspects of the two inputs, as shown in Figure 14.6 (Bowman
 7869 et al., 2016).

7870 Autoencoders can also be applied to longer texts, such as paragraphs and documents.
 7871 This enables applications such as **question answering**, which can be performed by match-
 7872 ing the encoding of the question with encodings of candidate answers (Miao et al., 2016).

7873 14.8.3 Supervised compositional methods

7874 Given a supervision signal, such as a label describing the sentiment or meaning of a sen-
 7875 tence, a wide range of compositional methods can be applied to compute a distributed
 7876 representation that then predicts the label. The simplest is to average the embeddings
 7877 of each word in the sentence, and pass this average through a feedforward neural net-
 7878 work (Iyyer et al., 2015). Convolutional and recurrent neural networks go further, with
 7879 the ability to effectively capturing multiword phenomena such as negation (Kalchbrenner
 7880 et al., 2014; Kim, 2014; Li et al., 2015; Tang et al., 2015). Another approach is to incorpo-
 7881 rate the syntactic structure of the sentence into a **recursive neural network**, in which the
 7882 representation for each syntactic constituent is computed from the representations of its
 7883 children (Socher et al., 2012). However, in many cases, recurrent neural networks perform
 7884 as well or better than recursive networks (Li et al., 2015).

7885 Whether convolutional, recurrent, or recursive, a key question is whether supervised
 7886 sentence representations are task-specific, or whether a single supervised sentence repre-
 7887 sentation model can yield useful performance on other tasks. Wieting et al. (2015) train a
 7888 variety of sentence embedding models for the task of labeling pairs of sentences as **para-**
 7889 **phrases**. They show that the resulting sentence embeddings give good performance for
 7890 sentiment analysis. The **Stanford Natural Language Inference corpus** classifies sentence
 7891 pairs as **entailments** (the truth of sentence i implies the truth of sentence j), **contradictions**
 7892 (the truth of sentence i implies the falsity of sentence j), and neutral (i neither entails nor
 7893 contradicts j). Sentence embeddings trained on this dataset transfer to a wide range of
 7894 classification tasks (Conneau et al., 2017).

7895 14.8.4 Hybrid distributed-symbolic representations

7896 The power of distributed representations is in their generality: the distributed represen-
 7897 tation of a unit of text can serve as a summary of its meaning, and therefore as the input
 7898 for downstream tasks such as classification, matching, and retrieval. For example, dis-
 7899 tributed sentence representations can be used to recognize the paraphrase relationship
 7900 between closely related sentences like the following:

7901 (14.5) Donald thanked Vlad profusely.

7902 (14.6) Donald conveyed to Vlad his profound appreciation.

7903 (14.7) Vlad was showered with gratitude by Donald.

7904 Symbolic representations are relatively brittle to this sort of variation, but are better
7905 suited to describe individual entities, the things that they do, and the things that are done
7906 to them. In examples (14.5)-(14.7), we not only know that somebody thanked someone
7907 else, but we can make a range of inferences about what has happened between the en-
7908 tities named *Donald* and *Vlad*. Because distributed representations do not treat entities
7909 symbolically, they lack the ability to reason about the roles played by entities across a sen-
7910 tence or larger discourse.⁵ A hybrid between distributed and symbolic representations
7911 might give the best of both worlds: robustness to the many different ways of describing
7912 the same event, plus the expressiveness to support inferences about entities and the roles
7913 that they play.

7914 A “top-down” hybrid approach is to begin with logical semantics (of the sort de-
7915 scribed in the previous two chapters), and but replace the predefined lexicon with a set
7916 of distributional word clusters (Poon and Domingos, 2009; Lewis and Steedman, 2013). A
7917 “bottom-up” approach is to add minimal symbolic structure to existing distributed repre-
7918 sentations, such as vector representations for each entity (Ji and Eisenstein, 2015; Wiseman
7919 et al., 2016). This has been shown to improve performance on two problems that we will
7920 encounter in the following chapters: classification of **discourse relations** between adj-
7921 cent sentences (chapter 16; Ji and Eisenstein, 2015), and **coreference resolution** of entity
7922 mentions (chapter 15; Wiseman et al., 2016; Ji et al., 2017). Research on hybrid seman-
7923 tic representations is still in an early stage, and future representations may deviate more
7924 boldly from existing symbolic and distributional approaches.

7925 Additional resources

7926 Turney and Pantel (2010) survey a number of facets of vector word representations, fo-
7927 cusing on matrix factorization methods. Schnabel et al. (2015) highlight problems with
7928 similarity-based evaluations of word embeddings, and present a novel evaluation that
7929 controls for word frequency. Baroni et al. (2014) address linguistic issues that arise in
7930 attempts to combine distributed and compositional representations.

7931 In bilingual and multilingual distributed representations, embeddings are estimated
7932 for translation pairs or tuples, such as (*dog*, *perro*, *chien*). These embeddings can improve
7933 machine translation (Zou et al., 2013; Klementiev et al., 2012), transfer natural language

⁵At a 2014 workshop on semantic parsing, this critique of distributed representations was expressed by Ray Mooney — a leading researcher in computational semantics — in a now well-known quote, “you can’t cram the meaning of a whole sentence into a single vector!”

7934 processing models across languages (Täckström et al., 2012), and make monolingual word
 7935 embeddings more accurate (Faruqui and Dyer, 2014). A typical approach is to learn a pro-
 7936 jection that maximizes the correlation of the distributed representations of each element
 7937 in a translation pair, which can be obtained from a bilingual dictionary. Distributed rep-
 7938 resentations can also be linked to perceptual information, such as image features. Bruni
 7939 et al. (2014) use textual descriptions of images to obtain visual contextual information for
 7940 various words, which supplements traditional distributional context. Image features can
 7941 also be inserted as contextual information in log bilinear language models (Kiros et al.,
 7942 2014), making it possible to automatically generate text descriptions of images.

7943 Exercises

- 7944 1. Prove that the sum of probabilities of paths through a hierarchical softmax tree is
 7945 equal to one.
2. In skipgram word embeddings, the negative sampling objective can be written as,

$$\mathcal{L} = \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{C}} \text{count}(i, j) \psi(i, j), \quad [14.29]$$

7946 with $\psi(i, j)$ is defined in Equation 14.23.

7947 Suppose we draw the negative samples from the empirical unigram distribution
 7948 $\hat{p}(i) = p_{\text{unigram}}(i)$. First, compute the expectation of \mathcal{L} with respect the negative
 7949 samples, using this probability.

7950 Next, take the derivative of this expectation with respect to the score of a single word
 7951 context pair $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$, and solve for the pointwise mutual information $\text{PMI}(i, j)$. You
 7952 should be able to show that at the optimum, the PMI is a simple function of $\sigma(\mathbf{u}_i \cdot \mathbf{v}_j)$
 7953 and the number of negative samples.

7954 (This exercise is part of a proof that shows that skipgram with negative sampling is
 7955 closely related to PMI-weighted matrix factorization.)

- 7956 3. * In Brown clustering, prove that the cluster merge that maximizes the average mu-
 7957 tual information (Equation 14.13) also maximizes the log-likelihood objective (Equa-
 7958 tion 14.12).
4. A simple way to compute a distributed phrase representation is to add up the dis-
 tributed representations of the words in the phrase. Consider a sentiment analysis
 model in which the predicted sentiment is, $\psi(\mathbf{w}) = \theta \cdot (\sum_{m=1}^M \mathbf{x}_m)$, where \mathbf{x}_m is
 the vector representation of word m . Prove that in such a model, the following two

inequalities cannot both hold:

$$\psi(\text{good}) > \psi(\text{not good}) \quad [14.30]$$

$$\psi(\text{bad}) < \psi(\text{not bad}). \quad [14.31]$$

7959 Then construct a similar example pair for the case in which phrase representations
 7960 are the *average* of the word representations.

5. Now let's consider a slight modification to the prediction model in the previous problem:

$$\psi(\mathbf{w}) = \boldsymbol{\theta} \cdot \text{ReLU}\left(\sum_{m=1}^M \mathbf{x}_m\right) \quad [14.32]$$

7961 Show that in this case, it *is* possible to achieve the inequalities above. Your solution
 7962 should provide the weights $\boldsymbol{\theta}$ and the embeddings \mathbf{x}_{good} , \mathbf{x}_{bad} , and \mathbf{x}_{not} .

7963 For the next two problems, download a set of pre-trained word embeddings, such as the
 7964 WORD2VEC or polyglot embeddings.

- 7965 6. Use cosine similarity to find the most similar words to: *dog*, *whale*, *before*, *however*,
 7966 *fabricate*.
- 7967 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
 7968 similarity.
- 7969 • *dog:puppy :: cat:?*
 7970 • *speak:speaker :: sing:?*
 7971 • *France:French :: England:?*
 7972 • *France:wine :: England:?*

7974 The remaining problems will require you to build a classifier and test its properties. Pick a
 7975 text classification dataset, such as the Cornell Movie Review data.⁶ Divide your data into
 7976 training (60%), development (20%), and test sets (20%), if no such division already exists.

- 7977 8. Train a convolutional neural network, with inputs set to pre-trained word embed-
 7978 dings from the previous two problems. Use an additional, fine-tuned embedding
 7979 for out-of-vocabulary words. Train until performance on the development set does
 7980 not improve. You can also use the development set to tune the model architecture,
 7981 such as the convolution width and depth. Report *F-MEASURE* and accuracy, as well
 7982 as training time.

⁶<http://www.cs.cornell.edu/people/pabo/movie-review-data/>

- 7983 9. Now modify your model from the previous problem to fine-tune the word embed-
7984 dings. Report F -MEASURE, accuracy, and training time.
- 7985 10. Try a simpler approach, in which word embeddings in the document are averaged,
7986 and then this average is passed through a feed-forward neural network. Again, use
7987 the development data to tune the model architecture. How close is the accuracy to
7988 the convolutional networks from the previous problems?

7989

Chapter 15

7990

Reference Resolution

7991 References are one of the most noticeable forms of linguistic ambiguity, afflicting not just
7992 automated natural language processing systems, but also fluent human readers. Warnings
7993 to avoid “ambiguous pronouns” are ubiquitous in manuals and tutorials on writing
7994 style. But referential ambiguity is not limited to pronouns, as shown in the text in Fig-
7995 ure 15.1. Each of the bracketed substrings refers to an entity that is introduced earlier
7996 in the passage. These references include the pronouns *he* and *his*, but also the shortened
7997 name *Cook*, and **nominals** such as *the firm* and *the firm’s biggest growth market*.

7998 **Reference resolution** subsumes several subtasks. This chapter will focus on **corefer-
7999 ence resolution**, which is the task of grouping spans of text that refer to a single underly-
8000 ing entity, or, in some cases, a single event: for example, the spans *Tim Cook*, *he*, and *Cook*
8001 are all **coreferent**. These individual spans are called **mentions**, because they mention an
8002 entity; the entity is sometimes called the **referent**. Each mention has a set of **antecedents**,
8003 which are preceding mentions that are coreferent; for the first mention of an entity, the an-
8004 tecedent set is empty. The task of **pronominal anaphora resolution** requires identifying
8005 only the antecedents of pronouns. In **entity linking**, references are resolved not to other
8006 spans of text, but to entities in a knowledge base. This task is discussed in chapter 17.

8007 Coreference resolution is a challenging problem for several reasons. Resolving differ-
8008 ent types of **referring expressions** requires different types of reasoning: the features and
8009 methods that are useful for resolving pronouns are different from those that are useful
8010 to resolve names and nominals. Coreference resolution involves not only linguistic rea-
8011 soning, but also world knowledge and pragmatics: you may not have known that China
8012 was Apple’s biggest growth market, but it is likely that you effortlessly resolved this ref-
8013 erence while reading the passage in Figure 15.1.¹ A further challenge is that coreference

¹This interpretation is based in part on the assumption that a **cooperative** author would not use the expression *the firm’s biggest growth market* to refer to an entity not yet mentioned in the article (Grice, 1975). **Pragmatics** is the discipline of linguistics concerned with the formalization of such assumptions (Huang,

- (15.1) *[[Apple Inc] Chief Executive Tim Cook] has jetted into [China] for talks with government officials as [he] seeks to clear up a pile of problems in [[the firm] 's biggest growth market] ... [Cook] is on [his] first trip to [the country] since taking over...*
-

Figure 15.1: Running example (Yee and Jones, 2012). Coreferring entity mentions are in brackets.

8014 resolution decisions are often entangled: each mention adds information about the entity,
 8015 which affects other coreference decisions. This means that coreference resolution must
 8016 be addressed as a structure prediction problem. But as we will see, there is no dynamic
 8017 program that allows the space of coreference decisions to be searched efficiently.

8018 **15.1 Forms of referring expressions**

8019 There are three main forms of referring expressions — pronouns, names, and nominals.

8020 **15.1.1 Pronouns**

8021 Pronouns are a closed class of words that are used for references. A natural way to think
 8022 about pronoun resolution is SMASH (Kehler, 2007):

- 8023 • Search for candidate antecedents;
 8024 • Match against hard agreement constraints;
 8025 • And Select using Heuristics, which are “soft” constraints such as recency, syntactic
 8026 prominence, and parallelism.

8027 **Search**

8028 In the search step, candidate antecedents are identified from the preceding text or speech.²
 8029 Any noun phrase can be a candidate antecedent, and pronoun resolution usually requires

2015).

²Pronouns whose referents come later are known as **cataphora**, as in the opening line from a novel by 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.

8030 parsing the text to identify all such noun phrases.³ Filtering heuristics can help to prune
 8031 the search space to noun phrases that are likely to be coreferent (Lee et al., 2013; Durrett
 8032 and Klein, 2013). In nested noun phrases, mentions are generally considered to be the
 8033 largest unit with a given **head word** (see § 10.5.2): thus, *Apple Inc. Chief Executive Tim Cook*
 8034 would be included as a mention, but *Tim Cook* would not, since they share the same head
 8035 word, *Cook*.

8036 **Matching constraints for pronouns**

8037 References and their antecedents must agree on semantic features such as number, person,
 8038 gender, and animacy. Consider the pronoun *he* in this passage from the running example:

8039 (15.2) Tim Cook has jetted in for talks with officials as [he] seeks to clear up a pile of
 8040 problems...

8041 The pronoun and possible antecedents have the following features:

- 8042 • *he*: singular, masculine, animate, third person
- 8043 • *officials*: plural, animate, third person
- 8044 • *talks*: plural, inanimate, third person
- 8045 • *Tim Cook*: singular, masculine, animate, third person

8046 The SMASH method searches backwards from *he*, discarding *officials* and *talks* because they
 8047 do not satisfy the agreements constraints.

8048 Another source of constraints comes from syntax — specifically, from the phrase struc-
 8049 ture trees discussed in chapter 10. Consider a parse tree in which both *x* and *y* are phrasal
 8050 constituents. The constituent *x* **c-commands** the constituent *y* iff the first branching node
 8051 above *x* also dominates *y*. For example, in Figure 15.2a, *Abigail* c-commands *her*, because
 8052 the first branching node above *Abigail*, *S*, also dominates *her*. Now, if *x* c-commands *y*,
 8053 **government and binding theory** (Chomsky, 1982) states that *y* can refer to *x* only if it is
 8054 a **reflexive pronoun** (e.g., *herself*). Furthermore, if *y* is a reflexive pronoun, then its an-
 8055 tecedent must c-command it. Thus, in Figure 15.2a, *her* cannot refer to *Abigail*; conversely,
 8056 if we replace *her* with *herself*, then the reflexive pronoun *must* refer to *Abigail*, since this is
 8057 the only candidate antecedent that c-commands it.

8058 Now consider the example shown in Figure 15.2b. Here, *Abigail* does not c-command
 8059 *her*, but *Abigail's mom* does. Thus, *her* can refer to *Abigail* — and we cannot use reflexive

³In the OntoNotes coreference annotations, verbs can also be antecedents, if they are later referenced by nominals (Pradhan et al., 2011):

(15.1) Sales of passenger cars [grew] 22%. [The strong growth] followed year-to-year increases.

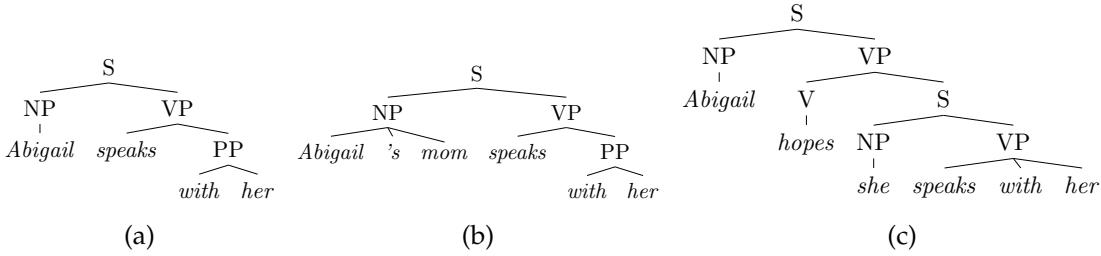


Figure 15.2: In (a), *Abigail* c-commands *her*; in (b), *Abigail* does not c-command *her*, but *Abigail's mom* does; in (c), the scope of *Abigail* is limited by the S non-terminal, so that *she* or *her* can bind to *Abigail*, but not both.

8060 *herself* in this context, unless we are talking about *Abigail's mom*. However, *her* does not
 8061 have to refer to *Abigail*. Finally, Figure 15.2c shows how these constraints are limited.
 8062 In this case, the pronoun *she* can refer to *Abigail*, because the S non-terminal puts *Abigail*
 8063 outside the domain of *she*. Similarly, *her* can also refer to *Abigail*. But *she* and *her* cannot be
 8064 coreferent, because *she* c-commands *her*.

8065 Heuristics

8066 After applying constraints, heuristics are applied to select among the remaining candidates.
 8067 Recency is a particularly strong heuristic. All things equal, readers will prefer
 8068 the more recent referent for a given pronoun, particularly when comparing referents that
 8069 occur in different sentences. Jurafsky and Martin (2009) offer the following example:

- 8070 (15.3) The doctor found an old map in the captain's chest. Jim found an even older map
 8071 hidden on the shelf. [It] described an island.

8072 Readers are expected to prefer the older map as the referent for the pronoun *it*.

8073 However, subjects are often preferred over objects, and this can contradict the preference
 8074 for recency when two candidate referents are in the same sentence. For example,

- 8075 (15.4) Asha loaned Mei a book on Spanish. [She] is always trying to help people.

8076 Here, we may prefer to link *she* to *Asha* rather than *Mei*, because of *Asha's* position in the
 8077 subject role of the preceding sentence. (Arguably, this preference would not be strong
 8078 enough to select *Asha* if the second sentence were *She is visiting Valencia next month*.)

8079 A third heuristic is parallelism:

- 8080 (15.5) Asha loaned Mei a book on Spanish. Olya loaned [her] a book on Portuguese.

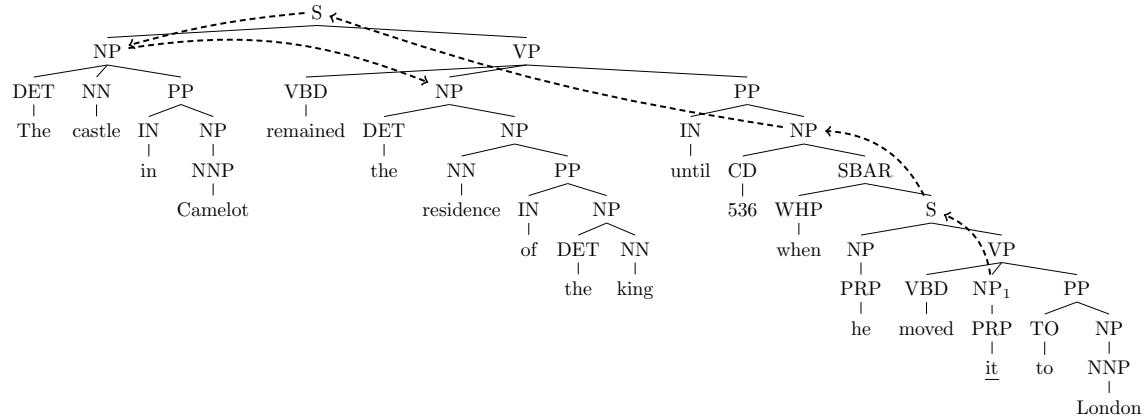


Figure 15.3: Left-to-right breadth-first tree traversal (Hobbs, 1978), indicating that the search for an antecedent for *it* (NP_1) would proceed in the following order: 536; *the castle in Camelot*; *the residence of the king*; *Camelot*; *the king*. Hobbs (1978) proposes semantic constraints to eliminate 536 and *the castle in Camelot* as candidates, since they are unlikely to be the direct object of the verb *move*.

8081 Here *Mei* is preferred as the referent for *her*, contradicting the preference for the subject
 8082 *Asha* in the preceding sentence.

8083 The recency and subject role heuristics can be unified by traversing the document in
 8084 a syntax-driven fashion (Hobbs, 1978): each preceding sentence is traversed breadth-first,
 8085 left-to-right (Figure 15.3). This heuristic successfully handles (15.4): *Asha* is preferred as
 8086 the referent for *she* because the subject NP is visited first. It also handles (15.3): the older
 8087 map is preferred as the referent for *it* because the more recent sentence is visited first. (An
 8088 alternative unification of recency and syntax is proposed by **centering theory** (Grosz et al.,
 8089 1995), which is discussed in detail in chapter 16.)

8090 In early work on reference resolution, the number of heuristics was small enough that
 8091 a set of numerical weights could be set by hand (Lappin and Leass, 1994). More recent
 8092 work uses machine learning to quantify the importance of each of these factors. However,
 8093 pronoun resolution cannot be completely solved by constraints and heuristics alone. This
 8094 is shown by the classic example pair (Winograd, 1972):

8095 (15.6) The [city council] denied [the protesters] a permit because [they] advocated / feared
 8096 violence.

8097 Without reasoning about the motivations of the city council and protesters, it is unlikely
 8098 that any system could correctly resolve both versions of this example.

8099 **Non-referential pronouns**

8100 While pronouns are generally used for reference, they need not refer to entities. The fol-
 8101 lowing examples show how pronouns can refer to propositions, events, and speech acts.

- 8102 (15.7) They told me that I was too ugly for show business, but I didn't believe [it].
 8103 (15.8) Asha saw Babak get angry, and I saw [it] too.
 8104 (15.9) Asha said she worked in security. I suppose [that]'s one way to put it.

8105 These forms of reference are generally not annotated in large-scale coreference resolution
 8106 datasets such as OntoNotes (Pradhan et al., 2011).

8107 Pronouns may also have **generic referents**:

- 8108 (15.10) A poor carpenter blames [her] tools.
 8109 (15.11) On the moon, [you] have to carry [your] own oxygen.
 8110 (15.12) Every farmer who owns a donkey beats [it]. (Geach, 1962)

8111 In the OntoNotes dataset, coreference is not annotated for generic referents, even in cases
 8112 like these examples, in which the same generic entity is mentioned multiple times.

8113 Some pronouns do not refer to anything at all:

- 8114 (15.13) *[It]'s raining.*
 [Il] pleut. (Fr)
 8115 (15.14) [It] 's money that she's really after.
 8116 (15.15) [It] is too bad that we have to work so hard.

8117 How can we automatically distinguish these usages of *it* from referential pronouns?
 8118 Consider the the difference between the following two examples (Bergsma et al., 2008):

- 8119 (15.16) You can make [it] in advance.
 8120 (15.17) You can make [it] in showbiz.

8121 In the second example, the pronoun *it* is non-referential. One way to see this is by substi-
 8122 tuting another pronoun, like *them*, into these examples:

- 8123 (15.18) You can make [them] in advance.
 8124 (15.19) ? You can make [them] in showbiz.

8125 The questionable grammaticality of the second example suggests that *it* is not referential.
 8126 Bergsma et al. (2008) operationalize this idea by comparing distributional statistics for the

8127 *n*-grams around the word *it*, testing how often other pronouns or nouns appear in the
8128 same context. In cases where nouns and other pronouns are infrequent, the *it* is unlikely
8129 to be referential.

8130 **15.1.2 Proper Nouns**

8131 If a proper noun is used as a referring expression, it often corefers with another proper
8132 noun, so that the coreference problem is simply to determine whether the two names
8133 match. Subsequent proper noun references often use a shortened form, as in the running
8134 example (Figure 15.1):

8135 (15.20) Apple Inc Chief Executive [Tim Cook] has jetted into China ... [Cook] is on his
8136 first business trip to the country ...

8137 A typical solution for proper noun coreference is to match the syntactic head words
8138 of the reference with the referent. In § 10.5.2, we saw that the head word of a phrase can
8139 be identified by applying head percolation rules to the phrasal parse tree; alternatively,
8140 the head can be identified as the root of the dependency subtree covering the name. For
8141 sequences of proper nouns, the head word will be the final token.

8142 There are a number of caveats to the practice of matching head words of proper nouns.

- 8143 • In the European tradition, family names tend to be more specific than given names,
8144 and family names usually come last. However, other traditions have other practices:
8145 for example, in Chinese names, the family name typically comes first; in Japanese,
8146 honorifics come after the name, as in *Nobu-San* (*Mr. Nobu*).
- 8147 • In organization names, the head word is often not the most informative, as in *Georgia*
8148 *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 modifiers (Lee et al., 2011).
- 8151 • Proper nouns can be nested, as in *[the CEO of [Microsoft]]*, resulting in head word
8152 match without coreference.

8153 Despite these difficulties, proper nouns are the easiest category of references to re-
8154 solve (Stoyanov et al., 2009). In machine learning systems, one solution is to include a
8155 range of matching features, including exact match, head match, and string inclusion. In
8156 addition to matching features, competitive systems (e.g., Bengtson and Roth, 2008) in-
8157 clude large lists, or **gazetteers**, of acronyms (e.g., *the National Basketball Association/NBA*),
8158 demonyms (e.g., *the Israelis/Israel*), and other aliases (e.g., *the Georgia Institute of Technology/Georgia Tech*).

8160 **15.1.3 Nominals**

8161 In coreference resolution, noun phrases that are neither pronouns nor proper nouns are
 8162 referred to as **nominals**. In the running example (Figure 15.1), nominal references include:
 8163 *the firm (Apple Inc); the firm's biggest growth market (China); and the country (China)*.

8164 Nominals are especially difficult to resolve (Denis and Baldridge, 2007; Durrett and
 8165 Klein, 2013), and the examples above suggest why this may be the case: world knowledge
 8166 is required to identify *Apple Inc* as a *firm*, and *China* as a *growth market*. Other difficult
 8167 examples include the use of colloquial expressions, such as coreference between *Clinton*
 8168 *campaign officials* and *the Clinton camp* (Soon et al., 2001).

8169 **15.2 Algorithms for coreference resolution**

The ground truth training data for coreference resolution is a set of mention sets, where all mentions within each set refer to a single entity.⁴ In the running example from Figure 15.1, the ground truth coreference annotation is:

$$c_1 = \{Apple\ Inc_{1:2}, the\ firm_{27:28}\} \quad [15.1]$$

$$c_2 = \{Apple\ Inc\ Chief\ Executive\ Tim\ Cook_{1:6}, he_{17}, Cook_{33}, his_{36}\} \quad [15.2]$$

$$c_3 = \{China_{10}, the\ firm\ 's\ biggest\ growth\ market_{27:32}, the\ country_{40:41}\} \quad [15.3]$$

8170 Each row specifies the token spans that mention an entity. (“Singleton” entities, which are
 8171 mentioned only once (e.g., *talks, government officials*), are excluded from the annotations.)
 8172 Equivalently, if given a set of M mentions, $\{m_i\}_{i=1}^M$, each mention i can be assigned to a
 8173 cluster z_i , where $z_i = z_j$ if i and j are coreferent. The cluster assignments z are invariant
 8174 under permutation. The unique clustering associated with the assignment z is written
 8175 $c(z)$.

8176 Coreference resolution can thus be viewed as a structure prediction problem, involving
 8177 two subtasks: identifying which spans of text mention entities, and then clustering
 8178 those spans.

8179 **Mention identification** The task of identifying mention spans for coreference resolution
 8180 is often performed by applying a set of heuristics to the phrase structure parse of each
 8181 sentence. A typical approach is to start with all noun phrases and named entities, and
 8182 then apply filtering rules to remove nested noun phrases with the same head (e.g., [*Apple*
 8183 *CEO [Tim Cook]*]), numeric entities (e.g., [*100 miles*], [*97%*]), non-referential *it*, etc (Lee

⁴In many annotations, the term **markable** is used to refer to spans of text that can *potentially* mention an entity. The set of markables includes non-referential pronouns, which does not mention any entity. Part of the job of the coreference system is to avoid incorrectly linking these non-referential markables to any mention chains.

et al., 2013; Durrett and Klein, 2013). In general, these deterministic approaches err in favor of recall, since the mention clustering component can choose to ignore false positive mentions, but cannot recover from false negatives. An alternative is to consider all spans (up to some finite length) as candidate mentions, performing mention identification and clustering jointly (Daumé III and Marcu, 2005; Lee et al., 2017).

Mention clustering The subtask of mention clustering will be the focus of the remainder of this chapter. There are two main classes of models. In *mention-based models*, the scoring function for a coreference clustering decomposes over pairs of mentions. These pairwise decisions are then aggregated, using a clustering heuristic. Mention-based coreference clustering can be treated as a fairly direct application of supervised classification or ranking. However, the mention-pair locality assumption can result in incoherent clusters, like $\{\text{Hillary Clinton} \leftarrow \text{Clinton} \leftarrow \text{Mr Clinton}\}$, in which the pairwise links score well, but the overall result is unsatisfactory. *Entity-based models* address this issue by scoring entities holistically. This can make inference more difficult, since the number of possible entity groupings is exponential in the number of mentions.

15.2.1 Mention-pair models

In the **mention-pair model**, a binary label $y_{i,j} \in \{0, 1\}$ is assigned to each pair of mentions (i, j) , where $i < j$. If i and j corefer ($z_i = z_j$), then $y_{i,j} = 1$; otherwise, $y_{i,j} = 0$. The mention *he* in Figure 15.1 is preceded by five other mentions: (1) *Apple Inc*; (2) *Apple Inc Chief Executive Tim Cook*; (3) *China*; (4) *talks*; (5) *government officials*. The correct mention pair labeling is $y_{2,6} = 1$ and $y_{i \neq 2,6} = 0$ for all other i . If a mention j introduces a new entity, such as mention 3 in the example, then $y_{i,j} = 0$ for all i . The same is true for “mentions” that do not refer to any entity, such as non-referential pronouns. If mention j refers to an entity that has been mentioned more than once, then $y_{i,j} = 1$ for all $i < j$ that mention the referent.

By transforming coreference into a set of binary labeling problems, the mention-pair model makes it possible to apply an off-the-shelf binary classifier (Soon et al., 2001). This classifier is applied to each mention j independently, searching backwards from j until finding an antecedent i which corefers with j with high confidence. After identifying a single **antecedent**, the remaining mention pair labels can be computed by transitivity: if $y_{i,j} = 1$ and $y_{j,k} = 1$, then $y_{i,k} = 1$.

Since the ground truth annotations give entity chains c but not individual mention-pair labels y , an additional heuristic must be employed to convert the labeled data into training examples for classification. A typical approach is to generate at most one positive labeled instance $y_{a_j,j} = 1$ for mention j , where a_j is the index of the most recent antecedent, $a_j = \max\{i : i < j \wedge z_i = z_j\}$. Negative labeled instances are generated for all for all $i \in \{a_j + 1, \dots, j\}$. In the running example, the most recent antecedent of the

8221 pronoun *he* is $a_6 = 2$, so the training data would be $y_{2,6} = 1$ and $y_{3,6} = y_{4,6} = y_{5,6} = 0$.
 8222 The variable $y_{1,6}$ is not part of the training data, because the first mention appears before
 8223 the true antecedent $a_6 = 2$.

8224 **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 ,

$$\hat{a}_i = \operatorname{argmax}_{a \in \{\epsilon, 1, 2, \dots, i-1\}} \psi_M(a, i), \quad [15.4]$$

8225 where $\psi_M(a, i)$ is a score for the mention pair (a, i) . If $a = \epsilon$, then mention i does not refer
 8226 to any previously-introduced entity — it is not **anaphoric**. Mention-ranking is similar to
 8227 the mention-pair model, but all candidates are considered simultaneously, and at most
 8228 a single antecedent is selected. The mention-ranking model explicitly accounts for the
 8229 possibility that mention i is not anaphoric, through the score $\psi_M(\epsilon, i)$. The determination
 8230 of anaphoricity can be made by a special classifier in a preprocessing step, so that non- ϵ
 8231 antecedents are identified only for spans that are determined to be anaphoric (Denis and
 8232 Baldridge, 2008).

8233 As a learning problem, ranking can be trained using the same objectives as in dis-
 8234 criminative classification. For each mention i , we can define a gold antecedent a_i^* , and an
 8235 associated loss, such as the hinge loss, $\ell_i = (1 - \psi_M(a_i^*, i) + \psi_M(\hat{a}, i))_+$ or the negative
 8236 log-likelihood, $\ell_i = -\log p(a_i^* | i; \theta)$. (For more on learning to rank, see § 17.1.1.) But as
 8237 with the mention-pair model, there is a mismatch between the labeled data, which comes
 8238 in the form of mention sets, and the desired supervision, which would indicate the spe-
 8239 cific antecedent of each mention. The antecedent variables $\{a_i\}_{i=1}^M$ relate to the mention
 8240 sets in a many-to-one mapping: each set of antecedents induces a single clustering, but a
 8241 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:

$$p(c | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} p(\mathbf{a} | \mathbf{m}) = \sum_{\mathbf{a} \in \mathcal{A}(c)} \prod_{i=1}^M p(a_i | i, \mathbf{m}) \quad [15.8]$$

$$p(a_i | i, \mathbf{m}) = \frac{\exp(\psi_M(a_i, i))}{\sum_{a' \in \{\epsilon, 1, 2, \dots, i-1\}} \exp(\psi_M(a', i))}. \quad [15.9]$$

8242 This objective rewards models that assign high scores for all valid antecedent structures.
 8243 In the running example, this would correspond to summing the probabilities of the two
 8244 valid antecedents for *Cook, he* and *Apple Inc Chief Executive Tim Cook*. In one of the exer-
 8245 cises, you will compute the number of valid antecedent structures for a given clustering.

8246 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:

$$m_1 = \text{Hillary Clinton} \quad [15.10]$$

$$m_2 = \text{Clinton} \quad [15.11]$$

$$m_3 = \text{Bill Clinton} \quad [15.12]$$

8247 A mention-pair system might predict $\hat{y}_{1,2} = 1, \hat{y}_{2,3} = 1, \hat{y}_{1,3} = 0$. Similarly, a mention-
 8248 ranking system might choose $\hat{a}_2 = 1$ and $\hat{a}_3 = 2$. Logically, if mentions 1 and 3 are both
 8249 coreferent with mention 2, then all three mentions must refer to the same entity. This
 8250 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 of n items is the **Bell number**,

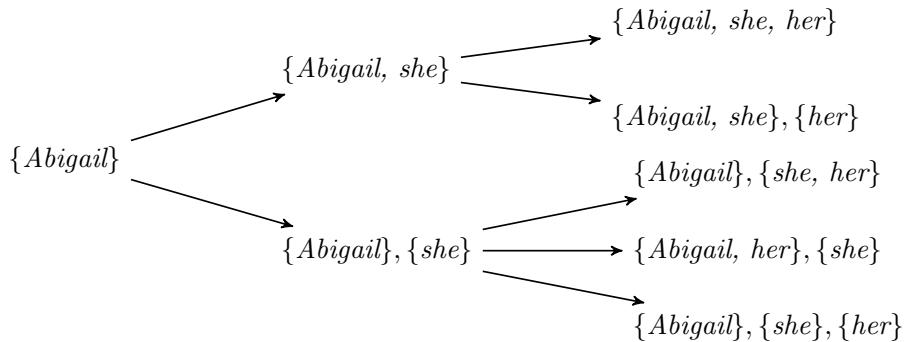


Figure 15.4: The Bell Tree for the sentence *Abigail hopes she speaks with her*. Which paths are excluded by the syntactic constraints mentioned in § 15.1.1?

which is defined by the following recurrence (Bell, 1934; Luo et al., 2004),

$$B_n = \sum_{k=0}^{n-1} B_k \binom{n-1}{k} B_0 = \quad B_1 = 1. \quad [15.14]$$

8268 This recurrence is illustrated by the Bell tree, which is applied to a short coreference prob-
 8269 lem in Figure 15.4. The Bell number B_n grows exponentially with n , making exhaustive
 8270 search of the space of clusterings impossible. For this reason, entity-based coreference
 8271 resolution typically involves incremental search, in which clustering decisions are based
 8272 on local evidence, in the hope of approximately optimizing the full objective in Equa-
 8273 tion 15.13. This approach is sometimes called **cluster ranking**, in contrast to mention
 8274 ranking.

8275 ***Generative models of coreference** Entity-based coreference can be approached through
 8276 probabilistic **generative models**, in which the mentions in the document are conditioned
 8277 on a set of latent entities (Haghghi and Klein, 2007, 2010). An advantage of these meth-
 8278 ods is that they can be learned from unlabeled data (Poon and Domingos, 2008, e.g.); a
 8279 disadvantage is that probabilistic inference is required not just for learning, but also for
 8280 prediction. Furthermore, generative models require independence assumptions that are
 8281 difficult to apply in coreference resolution, where the diverse and heterogeneous features
 8282 do not admit an easy decomposition into mutually independent subsets.

8283 Incremental cluster ranking

8284 The SMASH method (§ 15.1.1) can be extended to entity-based coreference resolution by
 8285 building up coreference clusters while moving through the document (Cardie and Wagstaff,
 8286 1999). At each mention, the algorithm iterates backwards through possible antecedent

clusters; but unlike SMASH, a cluster is selected only if *all* members of its cluster are compatible with the current mention. As mentions are added to a cluster, so are their features (e.g., gender, number, animacy). In this way, incoherent chains like *{Hillary Clinton, Clinton, Bill Clinton}* can be avoided. However, an incorrect assignment early in the document — a **search error** — might lead to a cascade of errors later on.

More sophisticated search strategies can help to ameliorate the risk of search errors. One approach is **beam search** (first discussed in § 11.3), in which a set of hypotheses is maintained throughout search. Each hypothesis represents a path through the Bell tree (Figure 15.4). Hypotheses are “expanded” either by adding the next mention to an existing cluster, or by starting a new cluster. Each expansion receives a score, based on Equation 15.13, and the top K hypotheses are kept on the beam as the algorithm moves to the next step.

Incremental cluster ranking can be made more accurate by performing multiple passes over the document, applying rules (or “sieves”) with increasing recall and decreasing precision at each pass (Lee et al., 2013). In the early passes, coreference links are proposed only between mentions that are highly likely to corefer (e.g., exact string match for full names and nominals). Information can then be shared among these mentions, so that when more permissive matching rules are applied later, agreement is preserved across the entire cluster. For example, in the case of *{Hillary Clinton, Clinton, she}*, the name-matching sieve would link *Clinton* and *Hillary Clinton*, and the pronoun-matching sieve would then link *she* to the combined cluster. A deterministic multi-pass system won nearly every track of the 2011 CoNLL shared task on coreference resolution (Pradhan et al., 2011). Given the dominance of machine learning in virtually all other areas of natural language processing — and more than fifteen years of prior work on machine learning for coreference — this was a surprising result, even if learning-based methods have subsequently regained the upper hand (e.g., Lee et al., 2018, the state of the art at the time of this writing).

8314 Incremental perceptron

Incremental coreference resolution can be learned with the **incremental perceptron**, as described in § 11.3.2. At mention i , each hypothesis on the beam corresponds to a clustering of mentions $1 \dots i - 1$, or equivalently, a path through the Bell tree up to position $i - 1$. As soon as none of the hypotheses on the beam are compatible with the gold coreference clustering, a perceptron update is made (Daumé III and Marcu, 2005). For concreteness, consider a linear cluster ranking model,

$$\psi_E(\{i : z_i = e\}) = \sum_{i:z_i=e} \boldsymbol{\theta} \cdot \mathbf{f}(i, \{j : j < i \wedge z_j = e\}), \quad [15.15]$$

where the score for each cluster is computed as the sum of scores of all mentions that are linked into the cluster, and $\mathbf{f}(i, \emptyset)$ is a set of features for the non-anaphoric mention that

8317 initiates the cluster.

8318 Using Figure 15.4 as an example, suppose that the ground truth is,

$$\mathbf{c}^* = \{\text{Abigail}, \text{her}\}, \{\text{she}\}, \quad [15.16]$$

8319 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:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \mathbf{f}(\mathbf{c}^*) - \mathbf{f}(\hat{\mathbf{c}}) \quad [15.18]$$

$$= \boldsymbol{\theta} + (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \emptyset)) - (\mathbf{f}(\text{Abigail}, \emptyset) + \mathbf{f}(\text{she}, \{\text{Abigail}\})) \quad [15.19]$$

$$= \boldsymbol{\theta} + \mathbf{f}(\text{she}, \emptyset) - \mathbf{f}(\text{she}, \{\text{Abigail}\}). \quad [15.20]$$

8320 This style of incremental update can also be applied to a margin loss between the gold
 8321 clustering and the top clustering on the beam. By backpropagating from this loss, it is also
 8322 possible to train a more complicated scoring function, such as a neural network in which
 8323 the score for each entity is a function of embeddings for the entity mentions (Wiseman
 8324 et al., 2015).

8325 Reinforcement learning

8326 **Reinforcement learning** is a topic worthy of a textbook of its own (Sutton and Barto,
 8327 1998),⁵ so this section will provide only a very brief overview, in the context of coreference
 8328 resolution. A stochastic **policy** assigns a probability to each possible **action**, conditional
 8329 on the context. The goal is to learn a policy that achieves a high expected reward, or
 8330 equivalently, a low expected cost.

8331 In incremental cluster ranking, a complete clustering on M mentions can be produced
 8332 by a sequence of M actions, in which the action z_i either merges mention i with an existing
 8333 cluster or begins a new cluster. We can therefore create a stochastic policy using the cluster
 8334 scores (Clark and Manning, 2016),

$$\Pr(z_i = e; \boldsymbol{\theta}) = \frac{\exp \psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})}{\sum_{e'} \exp \psi_E(i \cup \{j : z_j = e'\}; \boldsymbol{\theta})}, \quad [15.21]$$

8335 where $\psi_E(i \cup \{j : z_j = e\}; \boldsymbol{\theta})$ is the score under parameters $\boldsymbol{\theta}$ for assigning mention i to
 8336 cluster e . This score can be an arbitrary function of the mention i , the cluster e and its
 8337 (possibly empty) set of mentions; it can also include the history of actions taken thus far.

⁵A draft of the second edition can be found here: <http://incompleteideas.net/book/the-book-2nd.html>. Reinforcement learning has been used in spoken dialogue systems (Walker, 2000) and text-based game playing (Branavan et al., 2009), and was applied to coreference resolution by Clark and Manning (2015).

8338 If a policy assigns probability $p(c; \theta)$ to clustering c , then its expected loss is,

$$L(\theta) = \sum_{c \in \mathcal{C}(m)} p_\theta(c) \times \ell(c), \quad [15.22]$$

8339 where $\mathcal{C}(m)$ is the set of possible clusterings for mentions m . The loss $\ell(c)$ can be based on
 8340 any arbitrary scoring function, including the complex evaluation metrics used in corefer-
 8341 ence resolution (see § 15.4). This is an advantage of reinforcement learning, which can be
 8342 trained directly on the evaluation metric — unlike traditional supervised learning, which
 8343 requires a loss function that is differentiable and decomposable across individual deci-
 8344 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),

$$\frac{\partial}{\partial \theta} L(\theta) = E_{z \sim \mathcal{Z}(m)} \ell(c(z)) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i | z_{1:i-1}, m) \quad [15.23]$$

$$\approx \frac{1}{K} \sum_{k=1}^K \ell(c(z^{(k)})) \sum_{i=1}^M \frac{\partial}{\partial \theta} \log p(z_i^{(k)} | z_{1:i-1}^{(k)}, m), \quad [15.24]$$

8345 where each action sequence $z^{(k)}$ is sampled from the current policy. Unlike the incremen-
 8346 tal perceptron, an update is not made until the complete action sequence is available.

8347 Learning to search

8348 Policy gradient can suffer from high variance: while the average loss over K samples is
 8349 asymptotically equal to the expected reward of a given policy, this estimate may not be
 8350 accurate unless K is very large. This can make it difficult to allocate credit and blame to
 8351 individual actions. In **learning to search**, this problem is addressed through the addition
 8352 of an **oracle** policy, which is known to receive zero or small loss. The oracle policy can be
 8353 used in two ways:

- 8354 • The oracle can be used to generate partial hypotheses that are likely to score well,
 8355 by generating i actions from the initial state. These partial hypotheses are then used
 8356 as starting points for the learned policy. This is known as **roll-in**.

Algorithm 17 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  $\frac{\partial}{\partial \theta} L(\theta)$ 

```

- 8357 • The oracle can be used to compute the minimum possible loss from a given state, by
 8358 generating $M - i$ actions from the current state until completion. This is known as
 8359 **roll-out**.

8360 The oracle can be combined with the existing policy during both roll-in and roll-out, sam-
 8361 pling actions from each policy (Daumé III et al., 2009). One approach is to gradually
 8362 decrease the number of actions drawn from the oracle over the course of learning (Ross
 8363 et al., 2011).

8364 In the context of entity-based coreference resolution, Clark and Manning (2016) use
 8365 the learned policy for roll-in and the oracle policy for roll-out. Algorithm 17 shows how
 8366 the gradients on the policy weights are computed in this case. In this application, the
 8367 oracle is “noisy”, because it selects the action that minimizes only the *local* loss — the
 8368 accuracy of the coreference clustering up to mention i — rather than identifying the action
 8369 sequence that will lead to the best final coreference clustering on the entire document.
 8370 When learning from noisy oracles, it can be helpful to mix in actions from the current
 8371 policy with the oracle during roll-out (Chang et al., 2015).

8372 **15.3 Representations for coreference resolution**

8373 Historically, coreference resolution has employed an array of hand-engineered features
 8374 to capture the linguistic constraints and preferences described in § 15.1 (Soon et al., 2001).
 8375 Later work has documented the utility of lexical and bilexical features on mention pairs (Björkelund
 8376 and Nugues, 2011; Durrett and Klein, 2013). The most recent and successful methods re-
 8377 place many (but not all) of these features with distributed representations of mentions
 8378 and entities (Wiseman et al., 2015; Clark and Manning, 2016; Lee et al., 2017).

8379 **15.3.1 Features**

8380 Coreference features generally rely on a preprocessing pipeline to provide part-of-speech
 8381 tags and phrase structure parses. This pipeline makes it possible to design features that
 8382 capture many of the phenomena from § 15.1, and is also necessary for typical approaches
 8383 to mention identification. However, the pipeline may introduce errors that propagate
 8384 to the downstream coreference clustering system. Furthermore, the existence of such
 8385 a pipeline presupposes resources such as treebanks, which do not exist for many lan-
 8386 guages.⁶

8387 **Mention features**

8388 Features of individual mentions can help to predict anaphoricity. In systems where men-
 8389 tion detection is performed jointly with coreference resolution, these features can also
 8390 predict whether a span of text is likely to be a mention. For mention i , typical features
 8391 include:

8392 **Mention type.** Each span can be identified as a pronoun, name, or nominal, using the
 8393 part-of-speech of the head word of the mention: both the Penn Treebank and Uni-
 8394 versal Dependencies tagsets (§ 8.1.1) include tags for pronouns and proper nouns,
 8395 and all other heads can be marked as nominals (Haghghi and Klein, 2009).

8396 **Mention width.** The number of tokens in a mention is a rough predictor of its anaphor-
 8397 icity, with longer mentions being less likely to refer back to previously-defined enti-
 8398 ties.

8399 **Lexical features.** The first, last, and head words can help to predict anaphoricity; they are
 8400 also useful in conjunction with features such as mention type and part-of-speech,
 8401 providing a rough measure of agreement (Björkelund and Nugues, 2011). The num-
 8402 ber of lexical features can be very large, so it can be helpful to select only frequently-
 8403 occurring features (Durrett and Klein, 2013).

8404 **Morphosyntactic features.** These features include the part-of-speech, number, gender,
 8405 and dependency ancestors.

8406 The features for mention i and candidate antecedent a can be conjoined, producing
 8407 joint features that can help to assess the compatibility of the two mentions. For example,
 8408 Durrett and Klein (2013) conjoin each feature with the mention types of the anaphora
 8409 and the antecedent. Coreference resolution corpora such as ACE and OntoNotes contain

⁶The Universal Dependencies project has produced dependency treebanks for more than sixty languages. However, coreference features and mention detection are generally based on phrase structure trees, which exist for roughly two dozen languages. A list is available here: <https://en.wikipedia.org/wiki/Treebank>

8410 documents from various genres. By conjoining the genre with other features, it is possible
8411 to learn genre-specific feature weights.

8412 **Mention-pair features**

8413 For any pair of mentions i and j , typical features include:

8414 **Distance.** The number of intervening tokens, mentions, and sentences between i and j
8415 can all be used as distance features. These distances can be computed on the surface
8416 text, or on a transformed representation reflecting the breadth-first tree traversal
8417 (Figure 15.3). Rather than using the distances directly, they are typically binned,
8418 creating binary features.

8419 **String match.** A variety of string match features can be employed: exact match, suffix
8420 match, head match, and more complex matching rules that disregard irrelevant
8421 modifiers (Soon et al., 2001).

8422 **Compatibility.** Building on the model, features can measure the anaphor and antecedent
8423 agree with respect to morphosyntactic attributes such as gender, number, and ani-
8424 macy.

8425 **Nesting.** If one mention is nested inside another (e.g., *[The President of [France]]*), they
8426 generally cannot corefer.

8427 **Same speaker.** For documents with quotations, such as news articles, personal pronouns
8428 can be resolved only by determining the speaker for each mention (Lee et al., 2013).
8429 Coreference is also more likely between mentions from the same speaker.

8430 **Gazetteers.** These features indicate that the anaphor and candidate antecedent appear in
8431 a gazetteer of acronyms (e.g., *USA/United States*, *GATech/Georgia Tech*), demonymns
8432 (e.g., *Israel/Israeli*), or other aliases (e.g., *Knickerbockers/New York Knicks*).

8433 **Lexical semantics.** These features use a lexical resource such as WORDNET to determine
8434 whether the head words of the mentions are related through synonymy, antonymy,
8435 and hypernymy (§ 4.2).

8436 **Dependency paths.** The dependency path between the anaphor and candidate antecedent
8437 can help to determine whether the pair can corefer, under the government and bind-
8438 ing constraints described in § 15.1.1.

8439 Comprehensive lists of mention-pair features are offered by Bengtson and Roth (2008) and
8440 Rahman and Ng (2011). Neural network approaches use far fewer mention-pair features:
8441 for example, Lee et al. (2017) include only speaker, genre, distance, and mention width
8442 features.

8443 **Semantics** In many cases, coreference seems to require knowledge and semantic infer-
 8444 ences, as in the running example, where we link *China* with a *country* and a *growth market*. Some of this information can be gleaned from WORDNET, which defines a graph
 8445 over **synsets** (see § 4.2). For example, one of the synsets of *China* is an instance of an
 8446 Asian_nation#1, which in turn is a hyponym of country#2, a synset that includes
 8447 *country*.⁷ Such paths can be used to measure the similarity between concepts (Pedersen
 8448 et al., 2004), and this similarity can be incorporated into coreference resolution as a fea-
 8449 ture (Ponzetto and Strube, 2006). Similar ideas can be applied to knowledge graphs in-
 8450 duced from Wikipedia (Ponzetto and Strube, 2007). But while such approaches improve
 8451 relatively simple classification-based systems, they have proven less useful when added
 8452 to the current generation of techniques.⁸ For example, Durrett and Klein (2013) employ
 8453 a range of semantics-based features — WordNet synonymy and hypernymy relations on
 8454 head words, named entity types (e.g., person, organization), and unsupervised clustering
 8455 over nominal heads — but find that these features give minimal improvement over a
 8456 baseline system using surface features.

8458 **Entity features**

8459 Many of the features for entity-mention coreference are generated by aggregating mention-
 8460 pair features over all mentions in the candidate entity (Culotta et al., 2007; Rahman and
 8461 Ng, 2011). Specifically, for each binary mention-pair feature $f(i, j)$, we compute the fol-
 8462 lowing entity-mention features for mention i and entity $e = \{j : j < i \wedge z_j = e\}$.

- 8463 • ALL-TRUE: Feature $f(i, j)$ holds for all mentions $j \in e$.
- 8464 • MOST-TRUE: Feature $f(i, j)$ holds for at least half and fewer than all mentions $j \in e$.
- 8465 • MOST-FALSE: Feature $f(i, j)$ holds for at least one and fewer than half of all men-
 8466 tions $j \in e$.
- 8467 • NONE: Feature $f(i, j)$ does not hold for any mention $j \in e$.

8468 For scalar mention-pair features (e.g., distance features), aggregation can be performed by
 8469 computing the minimum, maximum, and median values across all mentions in the cluster.
 8470 Additional entity-mention features include the number of mentions currently clustered in
 8471 the entity, and ALL-X and MOST-X features for each mention type.

8472 **15.3.2 Distributed representations of mentions and entities**

8473 Recent work has emphasized distributed representations of both mentions and entities.
 8474 One potential advantage is that pre-trained embeddings could help to capture the se-

⁷teletype font is used to indicate wordnet synsets, and *italics* is used to indicate strings.

⁸This point was made by Michael Strube at a 2015 workshop, noting that as the quality of the machine learning models in coreference has improved, the benefit of including semantics has become negligible.

8475 mantic compatibility underlying nominal coreference, helping with difficult cases like
 8476 (*Apple, the firm*) and (*China, the firm's biggest growth market*). Furthermore, a distributed
 8477 representation of entities can be trained to capture semantic features that are added by
 8478 each mention.

8479 **Mention embeddings**

8480 Entity mentions can be embedded into a vector space, providing the base layer for neural
 8481 networks that score coreference decisions (Wiseman et al., 2015).

8482 **Constructing the mention embedding** Various approaches for embedding multiword
 8483 units can be applied (see § 14.8). Figure 15.5 shows a recurrent neural network approach,
 8484 which begins by running a bidirectional LSTM over the entire text, obtaining hidden states
 8485 from the left-to-right and right-to-left passes, $\mathbf{h}_m = [\overleftarrow{\mathbf{h}}_m; \overrightarrow{\mathbf{h}}_m]$. Each candidate mention
 8486 span (s, t) is then represented by the vertical concatenation of four vectors:

$$\mathbf{u}^{(s,t)} = [\mathbf{u}_{\text{first}}^{(s,t)}; \mathbf{u}_{\text{last}}^{(s,t)}; \mathbf{u}_{\text{head}}^{(s,t)}; \phi^{(s,t)}], \quad [15.25]$$

8487 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
 8488 embedding of the last word, $\mathbf{u}_{\text{head}}^{(s,t)}$ is the embedding of the “head” word, and $\phi^{(s,t)}$ is a
 8489 vector of surface features, such as the length of the span (Lee et al., 2017).

Attention over head words Rather than identifying the head word from the output of a parser, it can be computed from a neural **attention mechanism**:

$$\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m \quad [15.26]$$

$$\mathbf{a}^{(s,t)} = \text{SoftMax}([\tilde{\alpha}_{s+1}, \tilde{\alpha}_{s+2}, \dots, \tilde{\alpha}_t]) \quad [15.27]$$

$$\mathbf{u}_{\text{head}}^{(s,t)} = \sum_{m=s+1}^t a_m^{(s,t)} \mathbf{h}_m. \quad [15.28]$$

8490 Each token m gets a scalar score $\tilde{\alpha}_m = \theta_\alpha \cdot \mathbf{h}_m$, which is the dot product of the LSTM
 8491 hidden state \mathbf{h}_m and a vector of weights θ_α . The vector of scores for tokens in the span
 8492 $m \in \{s + 1, s + 2, \dots, t\}$ is then passed through a softmax layer, yielding a vector $\mathbf{a}^{(s,t)}$
 8493 that allocates one unit of attention across the span. This eliminates the need for syntactic
 8494 parsing to recover the head word; instead, the model learns to identify the most important
 8495 words in each span. Attention mechanisms were introduced in neural machine transla-
 8496 tion (Bahdanau et al., 2014), and are described in more detail in § 18.3.1.

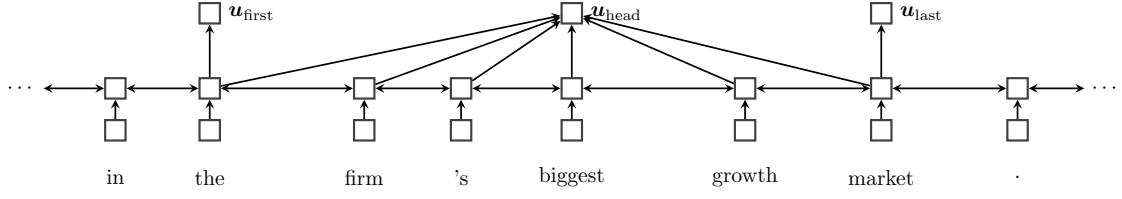


Figure 15.5: A bidirectional recurrent model of mention embeddings. The mention is represented by its first word, its last word, and an estimate of its head word, which is computed from a weighted average (Lee et al., 2017).

Using mention embeddings Given a set of mention embeddings, each mention i and candidate antecedent a is scored as,

$$\psi(a, i) = \psi_S(a) + \psi_S(i) + \psi_M(a, i) \quad [15.29]$$

$$\psi_S(a) = \text{FeedForward}_S(\mathbf{u}^{(a)}) \quad [15.30]$$

$$\psi_S(i) = \text{FeedForward}_S(\mathbf{u}^{(i)}) \quad [15.31]$$

$$\psi_M(a, i) = \text{FeedForward}_M([\mathbf{u}^{(a)}; \mathbf{u}^{(i)}; \mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}; \mathbf{f}(a, i, \mathbf{w})]), \quad [15.32]$$

where $\mathbf{u}^{(a)}$ and $\mathbf{u}^{(i)}$ are the embeddings for spans a and i respectively, as defined in Equation 15.25.

- The scores $\psi_S(a)$ quantify whether span a is likely to be a coreferring mention, independent of what it corefers with. This allows the model to learn identify mentions directly, rather than identifying mentions with a preprocessing step.
- The score $\psi_M(a, i)$ computes the compatibility of spans a and i . Its base layer is a vector that includes the embeddings of spans a and i , their elementwise product $\mathbf{u}^{(a)} \odot \mathbf{u}^{(i)}$, and a vector of surface features $\mathbf{f}(a, i, \mathbf{w})$, including distance, speaker, and genre information.

Lee et al. (2017) provide an error analysis that shows how this method can correctly link a *blaze* and a *fire*, while incorrectly linking *pilots* and *fight attendants*. In each case, the coreference decision is based on similarities in the word embeddings.

Rather than embedding individual mentions, Clark and Manning (2016) embed mention pairs. At the base layer, their network takes embeddings of the words in and around each mention, as well as one-hot vectors representing a few surface features, such as the distance and string matching features. This base layer is then passed through a multilayer feedforward network with ReLU nonlinearities, resulting in a representation of the mention pair. The output of the mention pair encoder $\mathbf{u}_{i,j}$ is used in the scoring function of a mention-ranking model, $\psi_M(i, j) = \theta \cdot \mathbf{u}_{i,j}$. A similar approach is used to score cluster

8516 pairs, constructing a cluster-pair encoding by **pooling** over the mention-pair encodings
8517 for all pairs of mentions within the two clusters.

8518 **Entity embeddings**

8519 In entity-based coreference resolution, each entity should be represented by properties of
8520 its mentions. In a distributed setting, we maintain a set of vector entity embeddings, v_e .
8521 Each candidate mention receives an embedding u_i ; Wiseman et al. (2016) compute this
8522 embedding by a single-layer neural network, applied to a vector of surface features. The
8523 decision of whether to merge mention i with entity e can then be driven by a feedforward
8524 network, $\psi_E(i, e) = \text{Feedforward}([v_e; u_i])$. If i is added to entity e , then its representa-
8525 tion is updated recurrently, $v_e \leftarrow f(v_e, u_i)$, using a recurrent neural network such as a
8526 long short-term memory (LSTM; chapter 6). Alternatively, we can apply a pooling oper-
8527 ation, such as max-pooling or average-pooling (chapter 3), setting $v_e \leftarrow \text{Pool}(v_e, u_i)$. In
8528 either case, the update to the representation of entity e can be thought of as adding new
8529 information about the entity from mention i .

8530 **15.4 Evaluating coreference resolution**

8531 The state of coreference evaluation is aggravatingly complex. Early attempts at sim-
8532 ple evaluation metrics were found to be susceptible to trivial baselines, such as placing
8533 each mention in its own cluster, or grouping all mentions into a single cluster. Follow-
8534 ing Denis and Baldridge (2009), the CoNLL 2011 shared task on coreference (Pradhan
8535 et al., 2011) formalized the practice of averaging across three different metrics: MUC (Vi-
8536 lain et al., 1995), B-CUBED (Bagga and Baldwin, 1998a), and CEAf (Luo, 2005). Refer-
8537 ence implementations of these metrics are available from Pradhan et al. (2014) at <https://github.com/conll/reference-coreference-scorers>.
8538

8539 **Additional resources**

8540 Ng (2010) surveys coreference resolution through 2010. Early work focused exclusively
8541 on pronoun resolution, with rule-based (Lappin and Leass, 1994) and probabilistic meth-
8542 ods (Ge et al., 1998). The full coreference resolution problem was popularized in a shared
8543 task associated with the sixth Message Understanding Conference, which included coref-
8544 erence annotations for training and test sets of thirty documents each (Grishman and
8545 Sundheim, 1996). An influential early paper was the decision tree approach of Soon et al.
8546 (2001), who introduced mention ranking. A comprehensive list of surface features for
8547 coreference resolution is offered by Bengtson and Roth (2008). Durrett and Klein (2013)
8548 improved on prior work by introducing a large lexicalized feature set; subsequent work
8549 has emphasized neural representations of entities and mentions (Wiseman et al., 2015).

8550 **Exercises**

8551 1. Select an article from today’s news, and annotate coreference for the first twenty
 8552 noun phrases that appear in the article (include nested noun phrases). Then specify
 8553 the mention-pair training data that would result from the first five noun phrases.

8554 2. Using your annotations from the preceding problem, compute the following statistics:
 8555

- 8556 • The number of times new entities are introduced by each of the three types of
 8557 referring expressions: pronouns, proper nouns, and nominals. Include “single-
 8558 ton” entities that are mentioned only once.
- 8559 • For each type of referring expression, compute the fraction of mentions that are
 8560 anaphoric.

8561 3. Apply a simple heuristic to all pronouns in the article from the previous exercise:
 8562 link each pronoun to the closest preceding noun phrase that agrees in gender, num-
 8563 ber, animacy, and person. Compute the following evaluation:

- 8564 • True positive: a pronoun that is linked to a noun phrase with which it is coref-
 8565 erent, or is labeled as the first mention of an entity when in fact it does not
 8566 corefer with any preceding mention. In this case, non-referential pronouns can
 8567 be true positives if they are marked as having no antecedent.
- 8568 • False positive: a pronoun that is linked to a noun phrase with which it is not
 8569 coreferent. This includes mistakenly linking singleton or non-referential pro-
 8570 nouns.
- 8571 • False negative: a pronoun that has at least one antecedent, but is either labeled
 8572 as not having an antecedent, or is linked to mention with which it does not
 8573 corefer.

8574 Compute the *F*-MEASURE for your method, and for a trivial baseline in which ev-
 8575 ery mention is its own entity. Are there any additional heuristics that would have
 8576 improved the performance of this method?

8577 4. Durrett and Klein (2013) compute the probability of the gold coreference clustering
 8578 by summing over all antecedent structures that are compatible with the clustering.
 8579 For example, if there are three mentions of a single entity, m_1, m_2, m_3 , there are two
 8580 possible antecedent structures: $a_2 = 1, a_3 = 1$ and $a_2 = 1, a_3 = 2$. Compute the
 8581 number of antecedent structures for a single entity with K mentions.

8582 5. Suppose that all mentions can be unambiguously divided into C classes, for exam-
 8583 ple by gender and number. Further suppose that mentions from different classes
 8584 can never corefer. In a document with M mentions, give upper and lower bounds

on the total number of possible coreference clusterings, in terms of the Bell numbers and the parameters M and C . Compute numerical upper and lower bounds for the case $M = 4, C = 2$.

6. Lee et al. (2017) propose a model that considers all contiguous spans in a document as possible mentions.

- 8590 a) In a document of length M , how many mention pairs must be evaluated? (All
8591 answers can be given in asymptotic, big-O notation.)
- 8592 b) To make inference more efficient, Lee et al. (2017) restrict consideration to spans
8593 of maximum length $L \ll M$. Under this restriction, how many mention pairs
8594 must be evaluated?
- 8595 c) To further improve inference, one might evaluate coreference only between
8596 pairs of mentions whose endpoints are separated by a maximum of D tokens.
8597 Under this additional restriction, how many mention pairs must be evaluated?

8598 7. In Spanish, the subject can be omitted when it is clear from context, e.g.,

8599 (15.21) *Las ballenas no son peces. Son mamíferos.*

The whales no are fish. Are mammals.

8600 Whales are not fish. They are mammals.

8601 Resolution of such **null subjects** is facilitated by the Spanish system of verb mor-
8602 phology, which includes distinctive suffixes for most combinations of person and
8603 number. For example, the verb form *son* ('are') agrees with the third-person plural
8604 pronouns *ellos* (masculine) and *ellas* (feminine), as well as the second-person plural
8605 *ustedes*.

8606 Suppose that you are given the following components:

- 8607 • A system that automatically identifies verbs with null subjects.
- 8608 • A function $c(j, p) \in \{0, 1\}$ that indicates whether pronoun p is compatible with
8609 null subject j , according to the verb morphology.
- 8610 • A trained mention-pair model, which computes scores $\psi(w_i, w_j, j - i) \in \mathbb{R}$ for
8611 all pairs of mentions i and j , scoring the pair by the antecedent mention w_i , the
8612 anaphor w_j , and the distance $j - i$.

8613 Describe an integer linear program that simultaneously performs two tasks: resolv-
8614 ing coreference among all entity mentions, and identifying suitable pronouns for all
8615 null subjects. In the example above, your program should link the null subject with
8616 *las ballenas* ('whales'), and identify *ellas* as the correct pronoun. For simplicity, you
8617 may assume that null subjects cannot be antecedents, and you need not worry about
8618 the transitivity constraint described in § 15.2.3.

8619 8. Use the policy gradient algorithm to compute the gradient for the following sce-
 8620 nario, based on the Bell tree in Figure 15.4:

- 8621 • The gold clustering c^* is $\{Abigail, her\}, \{she\}$.
 8621 • Drawing a single sequence of actions ($K = 1$) from the current policy, you
 obtain the following incremental clusterings:

$$\begin{aligned} c(a_1) &= \{Abigail\} \\ c(a_{1:2}) &= \{Abigail, she\} \\ c(a_{1:3}) &= \{Abigail, she\}, \{her\}. \end{aligned}$$

8622 • At each mention t , the space of actions A_t includes merging the mention with
 8623 each existing cluster or with the empty cluster. The probability of merging m_t
 8624 with cluster c is proportional to the exponentiated score for the merged cluster,

$$p(\text{Merge}(m_t, c)) \propto \exp \psi_E(m_t \cup c), \quad [15.33]$$

8625 where $\psi_E(m_t \cup c)$ is defined in Equation 15.15.

8626 Compute the gradient $\frac{\partial}{\partial \theta} L(\theta)$ in terms of the loss $\ell(c(a))$ and the features of each
 8627 (potential) cluster. Explain the differences between the gradient-based update $\theta \leftarrow \theta - \frac{\partial}{\partial \theta} L(\theta)$
 8628 and the incremental perceptron update from this same example.

8629 9. As discussed in § 15.1.1, some pronouns are not referential. In English, this occurs
 8630 frequently with the word *it*. Download the text of *Alice in Wonderland* from NLTK,
 8631 and examine the first ten appearances of *it*. For each occurrence:

- 8632 • First, examine a five-token window around the word. In the first example, this
 8633 window is,

8634 , but it had no

8635 Is there another pronoun that could be substituted for *it*? Consider *she*, *they*,
 8636 and *them*. In this case, both *she* and *they* yield grammatical substitutions. What
 8637 about the other ten appearances of *it*?

- 8638 • Now, view an fifteen-word window for each example. Based on this window,
 8639 mark whether you think the word *it* is referential.

8640 How often does the substitution test predict whether *it* is referential?

8641 10. Now try to automate the test, using the Google n -grams corpus (Brants and Franz,
 8642 2006). Specifically, find the count of each 5-gram containing *it*, and then compute
 8643 the counts of 5-grams in which *it* is replaced with other third-person pronouns: *he*,
 8644 *she*, *they*, *her*, *him*, *them*, *herself*, *himself*.

8645 There are various ways to get these counts. One approach is to download the
8646 raw data and search it; another is to construct web queries to [https://books.](https://books.google.com/ngrams)
8647 [google.com/ngrams](https://books.google.com/ngrams).

8648 Compare the ratio of the counts of the original 5-gram to the summed counts of
8649 the 5-grams created by substitution. Is this ratio a good predictor of whether *it* is
8650 referential?

8651 Chapter 16

8652 Discourse

8653 Applications of natural language processing often concern multi-sentence documents:
8654 from paragraph-long restaurant reviews, to 500-word newspaper articles, to 500-page
8655 novels. Yet most of the methods that we have discussed thus far are concerned with
8656 individual sentences. This chapter discusses theories and methods for handling multi-
8657 sentence linguistic phenomena, known collectively as **discourse**. There are diverse char-
8658 acterizations of discourse structure, and no single structure is ideal for every computa-
8659 tional application. This chapter covers some of the most well studied discourse repre-
8660 sentations, while highlighting computational models for identifying and exploiting these
8661 structures.

8662 16.1 Segments

8663 A document or conversation can be viewed as a sequence of **segments**, each of which is
8664 **cohesive** in its content and/or function. In Wikipedia biographies, these segments often
8665 pertain to various aspects to the subject's life: early years, major events, impact on others,
8666 and so on. This segmentation is organized around **topics**. Alternatively, scientific research
8667 articles are often organized by **functional themes**: the introduction, a survey of previous
8668 research, experimental setup, and results.

8669 Written texts often mark segments with section headers and related formatting de-
8670 vices. However, such formatting may be too coarse-grained to support applications such
8671 as the retrieval of specific passages of text that are relevant to a query (Hearst, 1997).
8672 Unformatted speech transcripts, such as meetings and lectures, are also an application
8673 scenario for segmentation (Carletta, 2007; Glass et al., 2007; Janin et al., 2003).

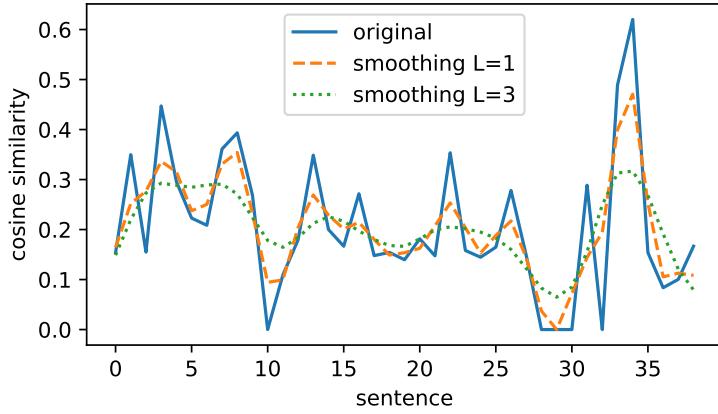


Figure 16.1: Smoothed cosine similarity among adjacent sentences in a news article. Local minima at $m = 10$ and $m = 29$ indicate likely segmentation points.

8674 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,

$$s_m = \frac{\mathbf{x}_m \cdot \mathbf{x}_{m+1}}{\|\mathbf{x}_m\|_2 \times \|\mathbf{x}_{m+1}\|_2} \quad [16.1]$$

$$\bar{s}_m = \sum_{\ell=0}^L k_\ell (s_{m+\ell} + s_{m-\ell}), \quad [16.2]$$

8675 with k_ℓ representing the value of a smoothing kernel of size L , e.g. $\mathbf{k} = [1, 0.5, 0.25]^\top$.
 8676 Segmentation points are then identified at local minima in the smoothed similarities \bar{s} ,
 8677 since these points indicate changes in the overall distribution of words in the text. An
 8678 example is shown in Figure 16.1.

8679 Text segmentation can also be formulated as a probabilistic model, in which each seg-
 8680 ment has a unique language model that defines the probability over the text in the seg-
 8681 ment (Utiyama and Isahara, 2001; Eisenstein and Barzilay, 2008; Du et al., 2013).¹ A good

¹There is a rich literature on how latent variable models (such as **latent Dirichlet allocation**) can track

8682 segmentation achieves high likelihood by grouping segments with similar word distribu-
8683 tions. This probabilistic approach can be extended to **hierarchical topic segmentation**, in
8684 which each topic segment is divided into subsegments (Eisenstein, 2009). All of these ap-
8685 proaches are unsupervised. While labeled data can be obtained from well-formatted texts
8686 such as textbooks, such annotations may not generalize to speech transcripts in alterna-
8687 tive domains. Supervised methods have been tried in cases where in-domain labeled data
8688 is available, substantially improving performance by learning weights on multiple types
8689 of features (Galley et al., 2003).

8690 16.1.2 Functional segmentation

8691 In some genres, there is a canonical set of communicative *functions*: for example, in sci-
8692 entific research articles, one such function is to communicate the general background for
8693 the article, another is to introduce a new contribution, or to describe the aim of the re-
8694 search (Teufel et al., 1999). A **functional segmentation** divides the document into con-
8695 tiguous segments, sometimes called **rhetorical zones**, in which each sentence has the same
8696 function. Teufel and Moens (2002) train a supervised classifier to identify the functional
8697 of each sentence in a set of scientific research articles, using features that describe the sen-
8698 tence's position in the text, its similarity to the rest of the article and title, tense and voice of
8699 the main verb, and the functional role of the previous sentence. Functional segmentation
8700 can also be performed without supervision. Noting that some types of Wikipedia arti-
8701 cles have very consistent functional segmentations (e.g., articles about cities or chemical
8702 elements), Chen et al. (2009) introduce an unsupervised model for functional segmenta-
8703 tion, which learns both the language model associated with each function and the typical
8704 patterning of functional segments across the article.

8705 16.2 Entities and reference

8706 Another dimension of discourse relates to which entities are mentioned throughout the
8707 text, and how. Consider the examples in Figure 16.2: Grosz et al. (1995) argue that the first
8708 discourse is more coherent. Do you agree? The examples differ in their choice of **refe-
8709 ring expressions** for the protagonist *John*, and in the syntactic constructions in sentences
8710 (b) and (d). The examples demonstrate the need for theoretical models to explain how
8711 referring expressions are chosen, and where they are placed within sentences. Such mod-
8712 els can then be used to help interpret the overall structure of the discourse, to measure
8713 discourse coherence, and to generate discourses in which referring expressions are used
8714 coherently.

topics across documents (Blei et al., 2003; Blei, 2012).

- | | |
|--|---|
| (16.1) a. John went to his favorite music store to buy a piano. b. He had frequented the store for many years. c. He was excited that he could finally buy a piano. d. He arrived just as the store was closing for the day | (16.2) a. John went to his favorite music store to buy a piano. b. It was a store John had frequented for many years. c. He was excited that he could finally buy a piano. d. It was closing just as John arrived. |
|--|---|

Figure 16.2: Two tellings of the same story (Grosz et al., 1995). The discourse on the left uses referring expressions coherently, while the one on the right does not.

8715 16.2.1 Centering theory

8716 **Centering theory** presents a unified account of the relationship between discourse struc-
8717 ture and entity reference (Grosz et al., 1995). According to the theory, every utterance in
8718 the discourse is characterized by a set of entities, known as *centers*.

- 8719 • The **forward-looking centers** in utterance m are all the entities that are mentioned
8720 in the utterance, $c_f(w_m) = \{e_1, e_2, \dots\}$. The forward-looking centers are partially
8721 ordered by their syntactic prominence, favoring subjects over objects, and objects
8722 over other positions (Brennan et al., 1987). For example, in example (1.1a) of Fig-
8723 ure 16.2, the ordered list of forward-looking centers in the first utterance is John, the
8724 music store, and the piano.
- 8725 • The **backward-looking center** $c_b(w_m)$ is the highest-ranked element in the set of
8726 forward-looking centers from the previous utterance $c_f(w_{m-1})$ that is also men-
8727 tioned in w_m . In example (1.1b) of item 16.1, the backward looking center is John.

8728 Given these two definitions, centering theory makes the following predictions about
8729 the form and position of referring expressions:

- 8730 1. If a pronoun appears in the utterance w_m , then the backward-looking center $c_b(w_m)$
8731 must also be realized as a pronoun. This rule argues against the use of *it* to refer
8732 to the piano store in Example (16.2d), since JOHN is the backward looking center of
8733 (16.2d), and he is mentioned by name and not by a pronoun.
- 8734 2. Sequences of utterances should retain the same backward-looking center if possible,
8735 and ideally, the backward-looking center should also be the top-ranked element in
8736 the list of forward-looking centers. This rule argues in favor of the preservation of
8737 JOHN as the backward-looking center throughout Example (16.1).

| | SKYLER | WALTER | DANGER | A GUY | THE DOOR |
|---------------------------------------|--------|--------|--------|-------|----------|
| You don't know who you're talking to, | S | - | - | - | - |
| so let me clue you in. | O | O | - | - | - |
| I am not in danger, Skyler. | X | S | X | - | - |
| I am the danger. | - | S | O | - | - |
| A guy opens his door and gets shot, | - | - | - | S | O |
| and you think that of me? | S | X | - | - | - |
| No. I am the one who knocks! | - | S | - | - | - |

Figure 16.3: The entity grid representation for a dialogue from the television show *Breaking Bad*.

8738 Centering theory unifies aspects of syntax, discourse, and anaphora resolution. However,
 8739 it can be difficult to clarify exactly how to rank the elements of each utterance, or even
 8740 how to partition a text or dialog into utterances (Poesio et al., 2004).

8741 16.2.2 The entity grid

8742 One way to formalize the ideas of centering theory is to arrange the entities in a text or
 8743 conversation in an **entity grid**. This is a data structure with one row per sentence, and
 8744 one column per entity (Barzilay and Lapata, 2008). Each cell $c(m, i)$ can take the following
 8745 values:

$$c(m, i) = \begin{cases} S, & \text{entity } i \text{ is in subject position in sentence } m \\ O, & \text{entity } i \text{ is in object position in sentence } m \\ X, & \text{entity } i \text{ appears in sentence } m, \text{ in neither subject nor object position} \\ -, & \text{entity } i \text{ does not appear in sentence } m. \end{cases} \quad [16.3]$$

8746 To populate the entity grid, syntactic parsing is applied to identify subject and object
 8747 positions, and coreference resolution is applied to link multiple mentions of a single entity.
 8748 An example is shown in Figure 16.3.

8749 After the grid is constructed, the coherence of a document can be measured by the
 8750 transitions between adjacent cells in each column. For example, the transition $(S \rightarrow S)$
 8751 keeps an entity in subject position across adjacent sentences; the transition $(O \rightarrow S)$ pro-
 8752 motes an entity from object position to subject position; the transition $(S \rightarrow -)$ drops the
 8753 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 text (Lapata, 2003), and disentangling multiple conversational threads in an online multi-party chat (Elsner and Charniak, 2010).

16.2.3 *Formal semantics beyond the sentence level

An alternative view of the role of entities in discourse focuses on formal semantics, and the construction of meaning representations for multi-sentence units. Consider the following two sentences (from Bird et al., 2009):

- (16.3) a. Angus owns a dog.
- b. It bit Irene.

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,

$$\exists x. \text{DOG}(x) \wedge \text{OWN}(\text{ANGUS}, x) \quad [16.5]$$

$$\text{BITE}(y, \text{IRENE}). \quad [16.6]$$

Unifying these two representations into the form of Equation 16.4 requires linking the unbound variable y from [16.6] with the quantified variable x in [16.5].² Discourse understanding therefore requires the reader to update a set of assignments, from variables to entities. This update would (presumably) link the *dog* in the first sentence of [16.3] with the unbound variable y in the second sentence, thereby licensing the conjunction in [16.4].³ This basic idea is at the root of **dynamic semantics** (Groenendijk and Stokhof, 1991). **Segmented discourse representation theory** links dynamic semantics with a set of **discourse relations**, which explain how adjacent units of text are rhetorically or conceptually related (Lascarides and Asher, 2007). The next section explores the theory of discourse relations in more detail.

²Groenendijk and Stokhof (1991) treats the y variable in Equation 16.6 as unbound. Even if it were bound locally with an existential quantifier ($\exists y \text{BITE}(y, \text{IRENE})$), the variable would still need to be reconciled with the quantified variable in Equation 16.5.

³This linking task is similar to coreference resolution (see chapter 15), but here the connections are between semantic variables, rather than spans of text.

8778 16.3 Relations

8779 In dependency grammar, sentences are characterized by a graph (usually a tree) of syntac-
8780 tic relations between words, such as NSUBJ and DET. A similar idea can be applied at the
8781 document level, identifying relations between discourse units, such as clauses, sentences,
8782 or paragraphs. The task of **discourse parsing** involves identifying discourse units and
8783 the relations that hold between them. These relations can then be applied to tasks such as
8784 document classification and summarization, as discussed in § 16.3.4.

8785 16.3.1 Shallow discourse relations

8786 The existence of discourse relations is hinted by **discourse connectives**, such as *however*,
8787 *moreover*, *meanwhile*, and *if ... then*. These connectives explicitly specify the relationship
8788 between adjacent units of text: *however* signals a contrastive relationship, *moreover* signals
8789 that the subsequent text elaborates or strengthens the point that was made immediately
8790 beforehand, *meanwhile* indicates that two events are contemporaneous, and *if ... then* sets
8791 up a conditional relationship. Discourse connectives can therefore be viewed as a starting
8792 point for the analysis of discourse relations.

8793 In **lexicalized tree-adjoining grammar for discourse (D-LTAG)**, each connective an-
8794 chors a relationship between two units of text (Webber, 2004). This model provides the
8795 theoretical basis for the **Penn Discourse Treebank (PDTB)**, the largest corpus of discourse
8796 relations in English (Prasad et al., 2008). It includes a hierarchical inventory of discourse
8797 relations (shown in Table 16.1), which is created by abstracting the meanings implied by
8798 the discourse connectives that appear in real texts (Knott, 1996). These relations are then
8799 annotated on the same corpus of news text used in the Penn Treebank (see § 9.2.2), adding
8800 the following information:

- 8801 • Each connective is annotated for the discourse relation or relations that it expresses,
8802 if any — many discourse connectives have senses in which they do not signal a
8803 discourse relation (Pitler and Nenkova, 2009).
- 8804 • For each discourse relation, the two arguments of the relation are specified as ARG1
8805 and ARG2, where ARG2 is constrained to be adjacent to the connective. These argu-
8806 ments may be sentences, but they may also smaller or larger units of text.
- 8807 • Adjacent sentences are annotated for **implicit discourse relations**, which are not
8808 marked by any connective. When a connective could be inserted between a pair
8809 of sentence, the annotator supplies it, and also labels its sense (e.g., example 16.5).
8810 In some cases, there is no relationship at all between a pair of adjacent sentences;
8811 in other cases, the only relation is that the adjacent sentences mention one or more
8812 shared entity. These phenomena are annotated as NOREL and ENTREL (entity rela-
8813 tion), respectively.

- 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.

8814 Examples of Penn Discourse Treebank annotations are shown in (16.4). In (16.4), the
 8815 word *therefore* acts as an explicit discourse connective, linking the two adjacent units of
 8816 text. The Treebank annotations also specify the “sense” of each relation, linking the con-
 8817 nective to a relation in the sense inventory shown in Table 16.1: in (16.4), the relation is
 8818 PRAGMATIC CAUSE:JUSTIFICATION because it relates to the author’s communicative in-
 8819 tentions. The word *therefore* can also signal causes in the external world (e.g., *He was*
 8820 *therefore forced to relinquish his plan*). In **discourse sense classification**, the goal is to de-
 8821 termine which discourse relation, if any, is expressed by each connective. A related task
 8822 is the classification of implicit discourse relations, as in (16.5). In this example, the re-
 8823 lationship between the adjacent sentences could be expressed by the connective *because*,
 8824 indicating a CAUSE:REASON relationship.

8825 **Classifying explicit discourse relations and their arguments**

8826 As suggested by the examples above, many connectives can be used to invoke multiple
 8827 types of discourse relations. Similarly, some connectives have senses that are unrelated
 8828 to discourse: for example, *and* functions as a discourse connective when it links propo-

- (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).

8829 sitions, but not when it links noun phrases (Lin et al., 2014). Nonetheless, the senses of
 8830 explicitly-marked discourse relations in the Penn Treebank are relatively easy to classify,
 8831 at least at the coarse-grained level. When classifying the four top-level PDTB relations,
 8832 90% accuracy can be obtained simply by selecting the most common relation for each
 8833 connective (Pitler and Nenkova, 2009). At the more fine-grained levels of the discourse
 8834 relation hierarchy, connectives are more ambiguous. This fact is reflected both in the ac-
 8835 curacy of automatic sense classification (Versley, 2011) and in interannotator agreement,
 8836 which falls to 80% for level-3 discourse relations (Prasad et al., 2008).

8837 A more challenging task for explicitly-marked discourse relations is to identify the
 8838 scope of the arguments. Discourse connectives need not be adjacent to ARG1, as shown
 8839 in item 16.6, where ARG1 follows ARG2; furthermore, the arguments need not be contigu-
 8840 ous, as shown in (16.7). For these reasons, recovering the arguments of each discourse
 8841 connective is a challenging subtask. Because intra-sentential arguments are often syn-
 8842 tactic constituents (see chapter 10), many approaches train a classifier to predict whether
 8843 each constituent is an appropriate argument for each explicit discourse connective (Well-
 8844 ner and Pustejovsky, 2007; Lin et al., 2014, e.g.).

8845 Classifying implicit discourse relations

Implicit discourse relations are considerably more difficult to classify and to annotate.⁴
 Most approaches are based on an encoding of each argument, which is then used as input

⁴In the dataset for the 2015 shared task on shallow discourse parsing, the interannotator agreement was 91% for explicit discourse relations and 81% for implicit relations, across all levels of detail (Xue et al., 2015).

to a nonlinear classifier:

$$\mathbf{z}^{(i)} = \text{Encode}(\mathbf{w}^{(i)}) \quad [16.7]$$

$$\mathbf{z}^{(i+1)} = \text{Encode}(\mathbf{w}^{(i+1)}) \quad [16.8]$$

$$\hat{y}_i = \underset{y}{\operatorname{argmax}} \Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}). \quad [16.9]$$

- 8846 This basic framework can be instantiated in several ways, including both feature-based
 8847 and neural encoders.

8848 **Feature-based approaches** Each argument can be encoded into a vector of surface fea-
 8849 tures. The encoding typically includes lexical features (all words, or all content words, or
 8850 a subset of words such as the first three and the main verb), Brown clusters of individ-
 8851 ual words (§ 14.4), and syntactic features such as terminal productions and dependency
 8852 arcs (Pitler et al., 2009; Lin et al., 2009; Rutherford and Xue, 2014). The classification func-
 8853 tion then has two parts. First, it creates a joint feature vector by combining the encodings
 8854 of each argument, typically by computing the cross-product of all features in each encod-
 8855 ing:

$$\mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \{(a \times b \times y) : (\mathbf{z}_a^{(i)} \mathbf{z}_b^{(i+1)})\} \quad [16.10]$$

8856 The size of this feature set grows with the square of the size of the vocabulary, so it can be
 8857 helpful to select a subset of features that are especially useful on the training data (Park
 8858 and Cardie, 2012). After \mathbf{f} is computed, any classifier can be trained to compute the final
 8859 score, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = \theta \cdot \mathbf{f}(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)})$.

8860 **Neural network approaches** In neural network architectures, the encoder is learned
 8861 jointly with the classifier as an end-to-end model. Each argument can be encoded using
 8862 a variety of neural architectures (surveyed in § 14.8): recursive (§ 10.6.1; Ji and Eisenstein,
 8863 2015), recurrent (§ 6.3; Ji et al., 2016), and convolutional (§ 3.4; Qin et al., 2017). The clas-
 8864 sification function can then be implemented as a feedforward neural network on the two
 8865 encodings (chapter 3; for examples, see Rutherford et al., 2017; Qin et al., 2017), or as a
 8866 simple bilinear product, $\Psi(y, \mathbf{z}^{(i)}, \mathbf{z}^{(i+1)}) = (\mathbf{z}^{(i)})^\top \Theta_y \mathbf{z}^{(i+1)}$ (Ji and Eisenstein, 2015). The
 8867 encoding model can be trained by backpropagation from the classification objective, such
 8868 as the margin loss. Rutherford et al. (2017) show that neural architectures outperform
 8869 feature-based approaches in most settings. While neural approaches require engineering
 8870 the network architecture (e.g., embedding size, number of hidden units in the classifier),
 8871 feature-based approaches also require significant engineering to incorporate linguistic re-
 8872 sources such as Brown clusters and parse trees, and to select a subset of relevant features.

8873 **16.3.2 Hierarchical discourse relations**

8874 In sentence parsing, adjacent phrases combine into larger constituents, ultimately pro-
 8875 ducing a single constituent for the entire sentence. The resulting tree structure enables
 8876 structured analysis of the sentence, with subtrees that represent syntactically coherent
 8877 chunks of meaning. **Rhetorical Structure Theory (RST)** extends this style of hierarchical
 8878 analysis to the discourse level (Mann and Thompson, 1988).

8879 The basic element of RST is the **discourse unit**, which refers to a contiguous span of
 8880 text. **Elementary discourse units** (EDUs) are the atomic elements in this framework, and
 8881 are typically (but not always) clauses.⁵ Each discourse relation combines two or more
 8882 adjacent discourse units into a larger, composite discourse unit; this process ultimately
 8883 unites the entire text into a tree-like structure.⁶

8884 **Nuclearity** In many discourse relations, one argument is primary. For example:

- 8885 (16.8) [LaShawn loves animals]_{*N*}
 8886 [She has nine dogs and one pig]_{*S*}

8887 In this example, the second sentence provides EVIDENCE for the point made in the first
 8888 sentence. The first sentence is thus the **nucleus** of the discourse relation, and the second
 8889 sentence is the **satellite**. The notion of nuclearity is similar to the head-modifier structure
 8890 of dependency parsing (see § 11.1.1). However, in RST, some relations have multiple
 8891 nuclei. For example, the arguments of the CONTRAST relation are equally important:

- 8892 (16.9) [The clash of ideologies survives this treatment]_{*N*}
 8893 [but the nuance and richness of Gorky's individual characters have vanished in the scuffle]_{*N*}⁷

8894 Relations that have multiple nuclei are called **coordinating**; relations with a single nu-
 8895 cleus are called **subordinating**. Subordinating relations are constrained to have only two
 8896 arguments, while coordinating relations (such as CONJUNCTION) may have more than
 8897 two.

⁵Details of discourse segmentation can be found in the RST annotation manual (Carlson and Marcu, 2001).

⁶While RST analyses are typically trees, this should not 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.

⁷from the RST Treebank (Carlson et al., 2002)

8898 **RST Relations** Rhetorical structure theory features a large inventory of discourse relations, which are divided into two high-level groups: subject matter relations, and presentational relations. Presentational relations are organized around the intended beliefs of the reader. For example, in (16.8), the second discourse unit provides evidence intended to increase the reader’s belief in the proposition expressed by the first discourse unit, that *LaShawn loves animals*. In contrast, subject-matter relations are meant to communicate additional facts about the propositions contained in the discourse units that they relate:

8905 (16.10) [the debt plan was rushed to completion]_N
 8906 [in order to be announced at the meeting]_S⁸

8907 In this example, the satellite describes a world state that is realized by the action described
 8908 in the nucleus. This relationship is about the world, and not about the author’s communicative intentions.

8910 **Example** Figure 16.5 depicts an RST analysis of a paragraph from a movie review. Asymmetric (subordinating) relations are depicted with an arrow from the satellite to the nucleus; symmetric (coordinating) relations are depicted with lines. The elementary discourse units 1F and 1G are combined into a larger discourse unit with the symmetric CONJUNCTION relation. The resulting discourse unit is then the satellite in a JUSTIFY relation with 1E.

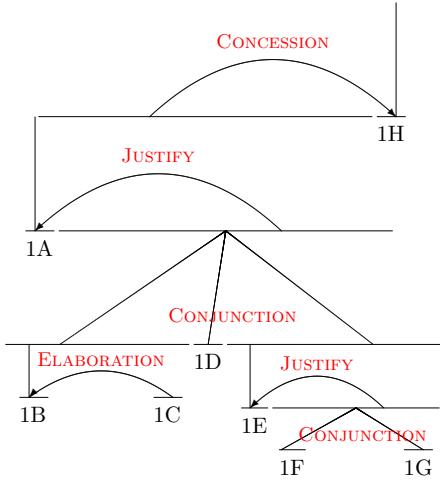
8916 Hierarchical discourse parsing

8917 The goal of discourse parsing is to recover a hierarchical structural analysis from a document text, such as the analysis in Figure 16.5. For now, let’s assume a segmentation of the document into elementary discourse units (EDUs); segmentation algorithms are discussed below. After segmentation, discourse parsing can be viewed as a combination of two components: the discourse relation classification techniques discussed in § 16.3.1, and algorithms for phrase-structure parsing, such as chart parsing and shift-reduce, which were discussed in chapter 10.

8924 Both chart parsing and shift-reduce require encoding composite discourse units, either in a discrete feature vector or a dense neural representation.⁹ Some discourse parsers
 8925 rely on the **strong compositionality criterion** (Marcu, 1996), which states the assumption
 8926 that a composite discourse unit can be represented by its nucleus. This criterion is used in
 8927 feature-based discourse parsing to determine the feature vector for a composite discourse
 8928 unit (Hernault et al., 2010); it is used in neural approaches to setting the vector encoding
 8930 for a composite discourse unit equal to the encoding of its nucleus (Ji and Eisenstein,

⁸from the RST Treebank (Carlson et al., 2002)

⁹To 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).



[It could have been a great movie]^{1A} [It does have beautiful scenery,]^{1B} [some of the best since Lord of the Rings.]^{1C} [The acting is well done.]^{1D} [and I really liked the son of the leader of the Samurai.]^{1E} [He was a likable chap.]^{1F} [and I hated to see him die.]^{1G} [But, other than all that, this movie is nothing more than hidden rip-offs.]^{1H}

Figure 16.5: A rhetorical structure theory analysis of a short movie review, adapted from Voll and Taboada (2007). Positive and negative sentiment words are underlined, indicating RST’s potential utility in document-level sentiment analysis.

8931 2014). An alternative neural approach is to learn a composition function over the compo-
 8932 nents of a composite discourse unit (Li et al., 2014), using a recursive neural network (see
 8933 § 14.8.3).

8934 **Bottom-up discourse parsing** Assume a segmentation of the text into N elementary
 8935 discourse units with base representations $\{z^{(i)}\}_{i=1}^N$, and assume a composition function
 8936 $\text{COMPOSE}(z^{(i)}, z^{(j)}, \ell)$, which maps two encodings and a discourse relation ℓ into a new
 8937 encoding. The composition function can follow the strong compositionality criterion and
 8938 simply select the encoding of the nucleus, or it can do something more complex. We
 8939 also need a scoring function $\Psi(z^{(i,k)}, z^{(k,j)}, \ell)$, which computes a scalar score for the (bi-
 8940 narized) discourse relation ℓ with left child covering the span $i + 1 : k$, and the right
 8941 child covering the span $k + 1 : j$. Given these components, we can construct vector rep-
 8942 resentations for each span, and this is the basic idea underlying **compositional vector**
 8943 **grammars** (Socher et al., 2013).

8944 These same components can also be used in bottom-up parsing, in a manner that is
 8945 similar to the CKY algorithm for weighted context-free grammars (see § 10.1): compute
 8946 the score and best analysis for each possible span of increasing lengths, while storing

8947 back-pointers that make it possible to recover the optimal parse of the entire input. How-
 8948 ever, there is an important distinction from CKY parsing: for each labeled span (i, j, ℓ) , we
 8949 must use the composition function to construct a representation $z^{(i,j,\ell)}$. This representa-
 8950 tion is then used to combine the discourse unit spanning $i + 1 : j$ in higher-level discourse
 8951 relations. The representation $z^{(i,j,\ell)}$ depends on the entire substructure of the unit span-
 8952 ning $i + 1 : j$, and this violates the locality assumption that underlie CKY’s optimality
 8953 guarantee. Bottom-up parsing with recursively constructed span representations is gen-
 8954 erally not guaranteed to find the best-scoring discourse parse. This problem is explored
 8955 in an exercise at the end of the chapter.

8956 **Transition-based discourse parsing** One drawback of bottom-up parsing is its cubic
 8957 time complexity in the length of the input. For long documents, transition-based parsing
 8958 is an appealing alternative. The shift-reduce algorithm (see § 10.6.2) can be applied to
 8959 discourse parsing fairly directly (Sagae, 2009): the stack stores a set of discourse units and
 8960 their representations, and each action is chosen by a function of these representations.
 8961 This function could be a linear product of weights and features, or it could be a neural
 8962 network applied to encodings of the discourse units. The REDUCE action then performs
 8963 composition on the two discourse units at the top of the stack, yielding a larger composite
 8964 discourse unit, which goes on top of the stack. All of the techniques for integrating learn-
 8965 ing and transition-based parsing, described in § 11.3, are applicable to discourse parsing.

8966 Segmenting discourse units

8967 In rhetorical structure theory, elementary discourse units do not cross the sentence bound-
 8968 ary, so discourse segmentation can be performed within sentences, assuming the sentence
 8969 segmentation is given. The segmentation of sentences into elementary discourse units is
 8970 typically performed using features of the syntactic analysis (Braud et al., 2017). One ap-
 8971 proach is to train a classifier to determine whether each syntactic constituent is an EDU,
 8972 using features such as the production, tree structure, and head words (Soricut and Marcu,
 8973 2003; Hernault et al., 2010). Another approach is to train a sequence labeling model, such
 8974 as a conditional random field (Sporleder and Lapata, 2005; Xuan Bach et al., 2012; Feng
 8975 et al., 2014). This is done using the BIO formalism for segmentation by sequence labeling,
 8976 described in § 8.3.

8977 16.3.3 Argumentation

8978 An alternative view of text-level relational structure focuses on **argumentation** (Stab and
 8979 Gurevych, 2014b). Each segment (typically a sentence or clause) may support or rebut
 8980 another segment, creating a graph structure over the text. In the following example (from
 8981 Peldszus and Stede, 2013), segment S_2 provides argumentative support for the proposi-
 8982 tion in the segment S_1 :

- 8983 (16.11) [We should tear the building down]_{S1}
8984 [because it is full of asbestos]_{S2}.

8985 Assertions may also support or rebut proposed links between two other assertions, cre-
8986 ating a **hypergraph**, which is a generalization of a graph to the case in which edges can
8987 join any number of vertices. This can be seen by introducing another sentence into the
8988 example:

- 8989 (16.12) [In principle it is possible to clean it up]_{S3}
8990 [but according to the mayor that is too expensive]_{S4}

8991 S3 acknowledges the validity of S2, but **undercuts** its support of S1. This can be repre-
8992 sented by introducing a hyperedge, $(S3, S2, S1)_{\text{undercut}}$, indicating that S3 undercuts the
8993 proposed relationship between S2 and S1. S4 then undercuts the relevance of S3.

8994 **Argumentation mining** is the task of recovering such structures from raw texts. At
8995 present, annotations of argumentation structure are relatively small: Stab and Gurevych
8996 (2014a) have annotated a collection of 90 persuasive essays, and Peldszus and Stede (2015)
8997 have solicited and annotated a set of 112 paragraph-length “microtexts” in German.

8998 16.3.4 Applications of discourse relations

8999 The predominant application of discourse parsing is to select content within a document.
9000 In rhetorical structure theory, the nucleus is considered the more important element of
9001 the relation, and is more likely to be part of a summary of the document; it may also
9002 be more informative for document classification. The D-LTAG theory that underlies the
9003 Penn Discourse Treebank lacks this notion of nuclearity, but arguments may have varying
9004 importance, depending on the relation type. For example, the span of text constituting
9005 ARG1 of an expansion relation is more likely to appear in a summary, while the sentence
9006 constituting ARG2 of an implicit relation is less likely (Louis et al., 2010). Discourse rela-
9007 tions may also signal segmentation points in the document structure. Explicit discourse
9008 markers have been shown to correlate with changes in subjectivity, and identifying such
9009 change points can improve document-level sentiment classification, by helping the clas-
9010 sifier to focus on the subjective parts of the text (Trivedi and Eisenstein, 2013; Yang and
9011 Cardie, 2014).

9012 Extractive Summarization

9013 Text **summarization** is the problem of converting a longer text into a shorter one, while
9014 still conveying the key facts, events, ideas, and sentiments from the original. In **extractive**
9015 **summarization**, the summary is a subset of the original text; in **abstractive summariza-**
9016 **tion**, the summary is produced *de novo*, by paraphrasing the original, or by first encoding

it into a semantic representation (see § 19.2). The main strategy for extractive summarization is to maximize coverage, choosing a subset of the document that best covers the concepts mentioned in the document as a whole; typically, coverage is approximated by bag-of-words overlap (Nenkova and McKeown, 2012). Coverage-based objectives can be supplemented by hierarchical discourse relations, using the principle of nuclearity: in any subordinating discourse relation, the nucleus is more critical to the overall meaning of the text, and is therefore more important to include in an extractive summary (Marcu, 1997a).¹⁰ This insight can be generalized from individual relations using the concept of **discourse depth** (Hirao et al., 2013): for each elementary discourse unit e , the discourse depth d_e is the number of relations in which a discourse unit containing e is the satellite.

Both discourse depth and nuclearity can be incorporated into extractive summarization, using constrained optimization. Let \mathbf{x}_n be a bag-of-words vector representation of elementary discourse unit n , let $y_n \in \{0, 1\}$ indicate whether n is included in the summary, and let d_n be the depth of unit n . Furthermore, let each discourse unit have a “head” h , which is defined recursively:

- if a discourse unit is produced by a subordinating relation, then its head is the head of the (unique) nucleus;
- if a discourse unit is produced by a coordinating relation, then its head is the head of the left-most nucleus;
- for each elementary discourse unit, its parent $\pi(n) \in \{\emptyset, 1, 2, \dots, N\}$ is the head of the smallest discourse unit containing n whose head is not n ;
- if n is the head of the discourse unit spanning the whole document, then $\pi(n) = \emptyset$.

With these definitions in place, discourse-driven extractive summarization can be formalized as (Hirao et al., 2013),

$$\begin{aligned} & \max_{\mathbf{y}=\{0,1\}^N} \sum_{n=1}^N y_n \frac{\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})}{d_n} \\ & \text{s.t. } \sum_{n=1}^N y_n \left(\sum_{j=1}^V x_{n,j} \right) \leq L \\ & \quad y_{\pi(n)} \geq y_n, \quad \forall n \text{ s.t. } \pi(n) \neq \emptyset \end{aligned} \tag{16.11}$$

where $\Psi(\mathbf{x}_n, \{\mathbf{x}_{1:N}\})$ measures the coverage of elementary discourse unit n with respect to the rest of the document, and $\sum_{j=1}^V x_{n,j}$ is the number of tokens in \mathbf{x}_n . The first constraint ensures that the number of tokens in the summary has an upper bound L . The

¹⁰Conversely, the arguments of a multi-nuclear relation should either both be included in the summary, or both excluded (Durrett et al., 2016).

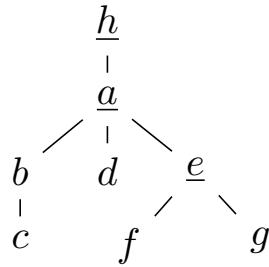


Figure 16.6: A discourse depth tree (Hirao et al., 2013) for the discourse parse from Figure 16.5, in which each elementary discourse unit is connected to its parent. The discourse units in one valid summary are underlined.

second constraint ensures that no elementary discourse unit is included unless its parent is also included. In this way, the discourse structure is used twice: to downweight the contributions of elementary discourse units that are not central to the discourse, and to ensure that the resulting structure is a subtree of the original discourse parse. The optimization problem in 16.11 can be solved with **integer linear programming**, described in § 13.2.2.¹¹

Figure 16.6 shows a discourse depth tree for the RST analysis from Figure 16.5, in which each elementary discourse is connected to (and below) its parent. The underlined discourse units in the figure constitute the following summary:

(16.13) It could have been a great movie, and I really liked the son of the leader of the Samurai. But, other than all that, this movie is nothing more than hidden rip-offs.

Document classification

Hierarchical discourse structures lend themselves naturally to text classification: in a subordinating discourse relation, the nucleus should play a stronger role in the classification decision than the satellite. Various implementations of this idea have been proposed.

- Focusing on within-sentence discourse relations and lexicon-based classification (see § 4.1.2), Voll and Taboada (2007) simply ignore the text in the satellites of each discourse relation.
- At the document level, elements of each discourse relation argument can be reweighted, favoring words in the nucleus, and disfavoring words in the satellite (Heerschap et al., 2011; Bhatia et al., 2015). This approach can be applied recursively, computing

¹¹Formally, 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).

weights across the entire document. The weights can be relation-specific, so that the features from the satellites of contrastive relations are discounted or even reversed.

- Alternatively, the hierarchical discourse structure can define the structure of a **recursive neural network** (see § 10.6.1). In this network, the representation of each discourse unit is computed from its arguments and from a parameter corresponding to the discourse relation (Ji and Smith, 2017).

Shallow, non-hierarchical discourse relations have also been applied to document classification. One approach is to impose a set of constraints on the analyses of individual discourse units, so that adjacent units have the same polarity when they are connected by a discourse relation indicating agreement, and opposite polarity when connected by a contrastive discourse relation, indicating disagreement (Somasundaran et al., 2009; Zirn et al., 2011). Yang and Cardie (2014) apply explicitly-marked relations from the Penn Discourse Treebank to the problem of sentence-level sentiment polarity classification (see § 4.1). They impose the following soft constraints:

- When a CONTRAST relation appears at the beginning of a sentence, the sentence should have the opposite sentiment polarity as its predecessor.
- When an EXPANSION or CONTINGENCY appears at the beginning of a sentence, it should have the same polarity as its predecessor.
- When a CONTRAST relation appears *within* a sentence, the sentence should have neutral polarity, since it is likely to express both sentiments.

These discourse-driven constraints are shown to improve performance on two datasets of product reviews.

9085 Coherence

Just as **grammaticality** is the property shared by well-structured sentences, **coherence** is the property shared by well-structured discourses. One application of discourse processing is to measure (and maximize) the coherence of computer-generated texts like translations and summaries (Kibble and Power, 2004). Coherence assessment is also used to evaluate human-generated texts, such as student essays (e.g., Miltsakaki and Kukich, 2004; Burstein et al., 2013).

Coherence subsumes a range of phenomena, many of which have been highlighted earlier in this chapter: e.g., that adjacent sentences should be lexically cohesive (Foltz et al., 1998; Ji et al., 2015; Li and Jurafsky, 2017), and that entity references should follow the principles of centering theory (Barzilay and Lapata, 2008; Nguyen and Joty, 2017). Discourse relations also bear on the coherence of a text in a variety of ways:

- Hierarchical discourse relations tend to have a “canonical ordering” of the nucleus and satellite (Mann and Thompson, 1988): for example, in the ELABORATION relation from rhetorical structure theory, the nucleus always comes first, while in the JUSTIFICATION relation, the satellite tends to be first (Marcu, 1997b).
- Discourse relations should be signaled by connectives that are appropriate to the semantic or functional relationship between the arguments: for example, a coherent text would be more likely to use *however* to signal a COMPARISON relation than a *temporal* relation (Kibble and Power, 2004).
- Discourse relations tend to appear in predictable sequences: for example, COMPARISON relations tend to immediately precede CONTINGENCY relations (Pitler et al., 2008). This observation can be formalized by generalizing the entity grid model (§ 16.2.2), so that each cell (i, j) provides information about the role of the discourse argument containing a mention of entity j in sentence i (Lin et al., 2011). For example, if the first sentence is ARG1 of a comparison relation, then any entity mentions in the sentence would be labeled COMP.ARG1. This approach can also be applied to RST discourse relations (Feng et al., 2014).

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.¹² 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).

¹²<http://homepages.inf.ed.ac.uk/mlap/coherence/>

9132 **Exercises**

- 9133 1. Some discourse connectives tend to occur between their arguments; others can pre-
9134 cede both arguments, and a few can follow both arguments. Indicate whether the
9135 following connectives can occur between, before, and after their arguments: *how-*
9136 *ever, but, while* (contrastive, not temporal), *although, therefore, nonetheless*.
 - 9137 2. This exercise is to be done in pairs. Each participant selects an article from to-
9138 day's news, and replaces all mentions of individual people with special tokens like
9139 PERSON1, PERSON2, and so on. The other participant should then use the rules
9140 of centering theory to guess each type of referring expression: full name (*Captain*
9141 *Ahab*), partial name (e.g., *Ahab*), nominal (e.g., *the ship's captain*), or pronoun. Check
9142 whether the predictions match the original text, and whether the text conforms to
9143 the rules of centering theory.
 - 9144 3. In this exercise, you will produce a figure similar to Figure 16.1.
 - 9145 a) Implement the smoothed cosine similarity metric from Equation 16.2, using the
9146 smoothing kernel $k = [.5, .3, .15, .05]$.
 - 9147 b) Download the text of a news article with at least ten paragraphs.
 - 9148 c) Compute and plot the smoothed similarity \bar{s} over the length of the article.
 - 9149 d) Identify *local minima* in \bar{s} as follows: first find all sentences m such that $\bar{s}_m <$
9150 $\bar{s}_{m \pm 1}$. Then search among these points to find the five sentences with the lowest
9151 \bar{s}_m .
 - 9152 e) How often do the five local minima correspond to paragraph boundaries?
 - 9153 • The fraction of local minima that are paragraph boundaries is the **precision-**
9154 **at- k** , where in this case, $k = 5$.
 - 9155 • The fraction of paragraph boundaries which are local minima is the **recall-**
9156 **at- k** .
 - 9157 • Compute precision-at- k and recall-at- k for $k = 3$ and $k = 10$.
 - 9158 4. One way to formulate text segmentation as a probabilistic model is through the use
9159 of the **Dirichlet Compound Multinomial** (DCM) distribution, which computes the
9160 probability of a bag-of-words, $DCM(\mathbf{x}; \boldsymbol{\alpha})$, where the parameter $\boldsymbol{\alpha}$ is a vector of
9161 positive reals. This distribution can be configured to assign high likelihood to bag-
9162 of-words vectors that are internally coherent, such that individual words appear re-
9163 peatedly: for example, this behavior can be observed for simple parameterizations,
9164 such as $\boldsymbol{\alpha} = \alpha \mathbf{1}$ with $\alpha < 1$.
- 9165 Let $\psi_{\boldsymbol{\alpha}}(i, j)$ represent the log-probability of a segment $w_{i+1:j}$ under a DCM distri-
9166 bution with parameter $\boldsymbol{\alpha}$. Give a dynamic program for segmenting a text into a total

of K segments maximizing the sum of log-probabilities $\sum_{k=1}^K \psi_\alpha(s_{k-1}, s_k)$, where s_k indexes the last token of segment k , and $s_0 = 0$. The time complexity of your dynamic program should not be worse than quadratic in the length of the input and linear in the number of segments.

5. Building on the previous problem, you will now adapt the CKY algorithm to perform hierarchical segmentation. Define a hierarchical segmentation as a set of segmentations $\{\{s_k^{(\ell)}\}_{k=1}^{K^{(\ell)}}\}_{\ell=1}^L$, where L is the segmentation depth. To ensure that the segmentation is hierarchically valid, we require that each segmentation point $s_k^{(\ell)}$ at level ℓ is also a segmentation point at level $\ell - 1$, where $\ell > 1$.

For simplicity, this problem focuses on binary hierarchical segmentation, so that each segment at level $\ell > 1$ has exactly 2 subsegments. Define the score of a hierarchical segmentation as the sum of the scores of all segments (at all levels), using the DCM log-probabilities from the previous problem as the segment scores. Give a CKY-like recurrence such that the optimal “parse” of the text is the maximum log-probability binary segmentation with exactly L levels.

6. The entity grid representation of centering theory can be used to compute a score for adjacent sentences, as described in § 16.2.2. Given a set of sentences, these scores can be used to compute an optimal ordering. Show that finding the ordering with the maximum log probability is NP-complete, by reduction from a well-known problem.
7. In § 16.3.2, it is noted that bottom-up parsing with compositional vector representations of each span is not guaranteed to be optimal. In this exercise, you will construct a minimal example proving this point. Consider a discourse with four units, with base representations $\{z^{(i)}\}_{i=1}^4$. Construct a scenario in which the parse selected by bottom-up parsing is not optimal, and give the precise mathematical conditions under which this suboptimal parse is selected. You may ignore the relation labels ℓ for the purpose of this example.
8. As noted in § 16.3.3, arguments can be described by hypergraphs, in which a segment may **undercut** a proposed edge between two other segments. Extend the model of extractive summarization described in § 16.3.4 to arguments, adding the following constraint: if segment i undercuts an argumentative relationship between j and k , then i cannot be included in the summary unless both j and k are included. Your solution should take the form of a set of *linear* constraints on an integer linear program — that is, each constraint can only involve addition and subtraction of variables.

In the next two exercises, you will explore the use of discourse connectives in a real corpus. Using NLTK, acquire the Brown corpus, and identify sentences that begin with any of the following connectives: *however, nevertheless, moreover, furthermore, thus*.

9204 9. Both lexical consistency and discourse connectives contribute to the **cohesion** of a
9205 text. We might therefore expect adjacent sentences that are joined by explicit dis-
9206 course connectives to also have higher word overlap. Using the Brown corpus, test
9207 this theory by computing the average cosine similarity between adjacent sentences
9208 that are connected by one of the connectives mentioned above. Compare this to the
9209 average cosine similarity of all other adjacent sentences. If you know how, perform
9210 a two-sample t-test to determine whether the observed difference is statistically sig-
9211 nificant.

9212 10. Group the above connectives into the following three discourse relations:

- 9213 • Expansion: *moreover, furthermore*
9214 • Comparison: *however, nevertheless*
9215 • Contingency: *thus*

9216 Focusing on pairs of sentences which are joined by one of these five connectives,
9217 build a classifier to predict the discourse relation from the text of the two adjacent
9218 sentences — taking care to ignore the connective itself. Use the first 30000 sentences
9219 of the Brown corpus as the training set, and the remaining sentences as the test
9220 set. Compare the performance of your classifier against simply choosing the most
9221 common class. Using a bag-of-words classifier, it is hard to do much better than this
9222 baseline, so consider more sophisticated alternatives!

9223

Part IV

9224

Applications

9225 Chapter 17

9226 Information extraction

9227 Computers offer powerful capabilities for searching and reasoning about structured records
9228 and relational data. Some have argued that the most important limitation of artificial in-
9229 telligence is not inference or learning, but simply having too little knowledge (Lenat et al.,
9230 1990). Natural language processing provides an appealing solution: automatically con-
9231 struct a structured **knowledge base** by reading natural language text.

9232 For example, many Wikipedia pages have an “infobox” that provides structured in-
9233 formation about an entity or event. An example is shown in Figure 17.1a: each row rep-
9234 resents one or more properties of the entity IN THE AEROPLANE OVER THE SEA, a record
9235 album. The set of properties is determined by a predefined **schema**, which applies to all
9236 record albums in Wikipedia. As shown in Figure 17.1b, the values for many of these fields
9237 are indicated directly in the first few sentences of text on the same Wikipedia page.

9238 The task of automatically constructing (or “populating”) an infobox from text is an
9239 example of **information extraction**. Much of information extraction can be described in
9240 terms of **entities**, **relations**, and **events**.

9241 • **Entities** are uniquely specified objects in the world, such as people (JEFF MANGUM),
9242 places (ATHENS, GEORGIA), organizations (MERGE RECORDS), and times (FEBRUARY
9243 10, 1998). Chapter 8 described the task of **named entity recognition**, which labels
9244 tokens as parts of entity spans. Now we will see how to go further, **linking** each
9245 entity **mention** to an element in a **knowledge base**.

- 9246 • **Relations** include a **predicate** and two **arguments**: for example, CAPITAL(GEORGIA, ATLANTA).
- **Events** involve multiple typed arguments. For example, the production and release

| Studio album by Neutral Milk Hotel | |
|------------------------------------|---------------------------------------|
| Released | February 10, 1998 |
| Recorded | July–September 1997 |
| Studio | Pet Sounds Studio, Denver, Colorado |
| Genre | Indie rock • psychedelic folk • lo-fi |
| Length | 39:55 |
| Label | Merge • Domino |
| Producer | Robert Schneider |

(a) A Wikipedia infobox

- (17.1) In the Aeroplane Over the Sea is the second and final studio album by the American indie rock band Neutral Milk Hotel.
- (17.2) It was released in the United States on February 10, 1998 on Merge Records and May 1998 on Blue Rose Records in the United Kingdom.
- (17.3) Jeff Mangum moved from Athens, Georgia to Denver, Colorado to prepare the bulk of the album's material with producer Robert Schneider, this time at Schneider's newly created Pet Sounds Studio at the home of Jim McIntyre.

- (b) The first few sentences of text. Strings that match fields or field names in the infobox are underlined; strings that mention other entities are wavy underlined.

Figure 17.1: From the Wikipedia page for the album “In the Aeroplane Over the Sea”, retrieved October 26, 2017.

of the album described in Figure 17.1 is described by the event,

```
<TITLE : IN THE AEROPLANE OVER THE SEA,
ARTIST : NEUTRAL MILK HOTEL,
RELEASE-DATE : 1998-FEB-10,...>
```

9247 The set of arguments for an event type is defined by a **schema**. Events often refer to
 9248 time-delimited occurrences: weddings, protests, purchases, terrorist attacks.

9249 Information extraction is similar to semantic role labeling (chapter 13): we may think
 9250 of predicates as corresponding to events, and the arguments as defining slots in the event
 9251 representation. However, the goals of information extraction are different. Rather than
 9252 accurately parsing every sentence, information extraction systems often focus on recog-
 9253 nizing a few key relation or event types, or on the task of identifying all properties of a
 9254 given entity. Information extraction is often evaluated by the correctness of the resulting
 9255 knowledge base, and not by how many sentences were accurately parsed. The goal is
 9256 sometimes described as **macro-reading**, as opposed to **micro-reading**, in which each sen-
 9257 tence must be analyzed correctly. Macro-reading systems are not penalized for ignoring
 9258 difficult sentences, as long as they can recover the same information from other, easier-
 9259 to-read sources. However, macro-reading systems must resolve apparent inconsistencies

9260 (was the album released on MERGE RECORDS or BLUE ROSE RECORDS?), requiring reasoning across the entire dataset.

9262 In addition to the basic tasks of recognizing entities, relations, and events, information
9263 extraction systems must handle negation, and must be able to distinguish statements of
9264 fact from hopes, fears, hunches, and hypotheticals. Finally, information extraction is often paired with the problem of **question answering**, which requires accurately parsing a
9266 query, and then selecting or generating a textual answer. Question answering systems can
9267 be built on knowledge bases that are extracted from large text corpora, or may attempt to
9268 identify answers directly from the source texts.

9269 17.1 Entities

9270 The starting point for information extraction is to identify mentions of entities in text.
9271 Consider the following example:

9272 (17.4) *The United States Army captured a hill overlooking Atlanta on May 14, 1864.*

9273 For this sentence, there are two goals:

- 9274 1. *Identify* the spans *United States Army*, *Atlanta*, and *May 14, 1864* as entity mentions.
9275 (The hill is not uniquely identified, so it is not a *named* entity.) We may also want to
9276 recognize the **named entity types**: organization, location, and date. This is **named**
9277 **entity recognition**, and is described in chapter 8.
- 9278 2. *Link* these spans to entities in a knowledge base: U.S. ARMY, ATLANTA, and 1864-
9279 MAY-14. This task is known as **entity linking**.

9280 The strings to be linked to entities are **mentions** — similar to the use of this term in
9281 coreference resolution. In some formulations of the entity linking task, only named entities
9282 are candidates for linking. This is sometimes called **named entity linking** (Ling et al.,
9283 2015). In other formulations, such as **Wikification** (Milne and Witten, 2008), any string
9284 can be a mention. The set of target entities often corresponds to Wikipedia pages, and
9285 Wikipedia is the basis for more comprehensive knowledge bases such as YAGO (Suchanek
9286 et al., 2007), DBPedia (Auer et al., 2007), and Freebase (Bollacker et al., 2008). Entity link-
9287 ing may also be performed in more “closed” settings, where a much smaller list of targets
9288 is provided in advance. The system must also determine if a mention does not refer to
9289 any entity in the knowledge base, sometimes called a **NIL entity** (McNamee and Dang,
9290 2009).

9291 Returning to (17.4), the three entity mentions may seem unambiguous. But the Wikipedia
9292 disambiguation page for the string *Atlanta* says otherwise:¹ there are more than twenty

¹[https://en.wikipedia.org/wiki/Atlanta_\(disambiguation\)](https://en.wikipedia.org/wiki/Atlanta_(disambiguation)), retrieved November 1, 2017.

9293 different towns and cities, five United States Navy vessels, a magazine, a television show,
 9294 a band, and a singer — each prominent enough to have its own Wikipedia page. We now
 9295 consider how to choose among these dozens of possibilities. In this chapter we will focus
 9296 on supervised approaches. Unsupervised entity linking is closely related to the problem
 9297 of **cross-document coreference resolution**, where the task is to identify pairs of mentions
 9298 that corefer, across document boundaries (Bagga and Baldwin, 1998b; Singh et al., 2011).

9299 17.1.1 Entity linking by learning to rank

9300 Entity linking is often formulated as a **ranking** problem,

$$\hat{y} = \operatorname{argmax}_{y \in \mathcal{Y}(x)} \Psi(y, x, c), \quad [17.1]$$

9301 where y is a target entity, x is a description of the mention, $\mathcal{Y}(x)$ is a set of candidate
 9302 entities, and c is a description of the context — such as the other text in the document,
 9303 or its metadata. The function Ψ is a scoring function, which could be a linear model,
 9304 $\Psi(y, x, c) = \theta \cdot f(y, x, c)$, or a more complex function such as a neural network. In either
 9305 case, the scoring function can be learned by minimizing a margin-based **ranking loss**,

$$\ell(\hat{y}, y^{(i)}, x^{(i)}, c^{(i)}) = (\Psi(\hat{y}, x^{(i)}, c^{(i)}) - \Psi(y^{(i)}, x^{(i)}, c^{(i)}) + 1)_+, \quad [17.2]$$

9306 where $y^{(i)}$ is the ground truth and $\hat{y} \neq y^{(i)}$ is the predicted target for mention $x^{(i)}$ in
 9307 context $c^{(i)}$ (Joachims, 2002; Dredze et al., 2010).

9308 **Candidate identification** For computational tractability, it is helpful to restrict the set of
 9309 candidates, $\mathcal{Y}(x)$. One approach is to use a **name dictionary**, which maps from strings
 9310 to the entities that they might mention. This mapping is many-to-many: a string such as
 9311 *Atlanta* can refer to multiple entities, and conversely, an entity such as ATLANTA can be
 9312 referenced by multiple strings. A name dictionary can be extracted from Wikipedia, with
 9313 links between each Wikipedia entity page and the anchor text of all hyperlinks that point
 9314 to the page (Bunescu and Pasca, 2006; Ratinov et al., 2011). To improve recall, the name
 9315 dictionary can be augmented by partial and approximate matching (Dredze et al., 2010),
 9316 but as the set of candidates grows, the risk of false positives increases. For example, the
 9317 string *Atlanta* is a partial match to *the Atlanta Fed* (a name for the FEDERAL RESERVE BANK
 9318 OF ATLANTA), and a noisy match (edit distance of one) from *Atalanta* (a heroine in Greek
 9319 mythology and an Italian soccer team).

9320 **Features** Feature-based approaches to entity ranking rely on three main types of local
 9321 information (Dredze et al., 2010):

- The similarity of the mention string to the canonical entity name, as quantified by string similarity. This feature would elevate the city ATLANTA over the basketball team ATLANTA HAWKS for the string *Atlanta*.
- The popularity of the entity, which can be measured by Wikipedia page views or PageRank in the Wikipedia link graph. This feature would elevate ATLANTA, GEORGIA over the unincorporated community of ATLANTA, OHIO.
- The entity type, as output by the named entity recognition system. This feature would elevate the city of ATLANTA over the magazine ATLANTA in contexts where the mention is tagged as a location.

In addition to these local features, the document context can also help. If *Jamaica* is mentioned in a document about the Caribbean, it is likely to refer to the island nation; in the context of New York, it is likely to refer to the neighborhood in Queens; in the context of a menu, it might refer to a hibiscus tea beverage. Such hints can be formalized by computing the similarity between the Wikipedia page describing each candidate entity and the mention context $c^{(i)}$, which may include the bag-of-words representing the document (Dredze et al., 2010; Hoffart et al., 2011) or a smaller window of text around the mention (Ratinov et al., 2011). For example, we can compute the cosine similarity between bag-of-words vectors for the context and entity description, typically weighted using **inverse document frequency** to emphasize rare words.²

Neural entity linking An alternative approach is to compute the score for each entity candidate using distributed vector representations of the entities, mentions, and context. For example, for the task of entity linking in Twitter, Yang et al. (2016) employ the bilinear scoring function,

$$\Psi(y, x, c) = v_y^\top \Theta^{(y,x)} x + v_y^\top \Theta^{(y,c)} c, \quad [17.3]$$

with $v_y \in \mathbb{R}^{K_y}$ as the vector embedding of entity y , $x \in \mathbb{R}^{K_x}$ as the embedding of the mention, $c \in \mathbb{R}^{K_c}$ as the embedding of the context, and the matrices $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ as parameters that score the compatibility of each entity with respect to the mention and context. Each of the vector embeddings can be learned from an end-to-end objective, or pre-trained on unlabeled data.

- Pretrained **entity embeddings** can be obtained from an existing knowledge base (Bordes et al., 2011, 2013), or by running a word embedding algorithm such as WORD2VEC

²The **document frequency** of word j is $DF(j) = \frac{1}{N} \sum_{i=1}^N \delta(x_j^{(i)} > 0)$, equal to the number of documents in which the word appears. The contribution of each word to the cosine similarity of two bag-of-words vectors can be weighted by the **inverse document frequency** $\frac{1}{DF(j)}$ or $\log \frac{1}{DF(j)}$, to emphasize rare words (Spärck Jones, 1972).

- 9352 on the text of Wikipedia, with hyperlinks substituted for the anchor text.³
- 9353 • The embedding of the mention x can be computed by averaging the embeddings
 9354 of the words in the mention (Yang et al., 2016), or by the compositional techniques
 9355 described in § 14.8.
- 9356 • The embedding of the context c can also be computed from the embeddings of the
 9357 words in the context. A **denoising autoencoder** learns a function from raw text to
 9358 dense K -dimensional vector encodings by minimizing a reconstruction loss (Vin-
 9359 cent et al., 2010),

$$\min_{\theta_g, \theta_h} \sum_{i=1}^N \|\mathbf{x}^{(i)} - g(h(\tilde{\mathbf{x}}^{(i)}; \theta_h); \theta_g)\|^2, \quad [17.4]$$

9360 where $\tilde{\mathbf{x}}^{(i)}$ is a noisy version of the bag-of-words counts $\mathbf{x}^{(i)}$, which is produced by
 9361 randomly setting some counts to zero; $h : \mathbb{R}^V \rightarrow \mathbb{R}^K$ is an encoder with parameters
 9362 θ_h ; and $g : \mathbb{R}^K \rightarrow \mathbb{R}^V$, with parameters θ_g . The encoder and decoder functions
 9363 are typically implemented as feedforward neural networks. To apply this model to
 9364 entity linking, each entity and context are initially represented by the encoding of
 9365 their bag-of-words vectors, $h(e)$ and $g(c)$, and these encodings are then fine-tuned
 9366 from labeled data (He et al., 2013). The context vector c can also be obtained by
 9367 convolution (§ 3.4) on the embeddings of words in the document (Sun et al., 2015),
 9368 or by examining metadata such as the author’s social network (Yang et al., 2016).

9369 The remaining parameters $\Theta^{(y,x)}$ and $\Theta^{(y,c)}$ can be trained by backpropagation from the
 9370 margin loss in Equation 17.2.

9367 17.1.2 Collective entity linking

9371 Entity linking can be more accurate when it is performed jointly across a document. To
 9372 see why, consider the following lists:

- 9373 (17.5) California, Oregon, Washington
 9374 (17.6) Baltimore, Washington, Philadelphia
 9375 (17.7) Washington, Adams, Jefferson

9376 In each case, the term *Washington* refers to a different entity, and this reference is strongly
 9377 suggested by the other entries on the list. In the last list, all three names are highly am-
 9378 biguous — there are dozens of other *Adams* and *Jefferson* entities in Wikipedia. But a

³Pre-trained entity embeddings can be downloaded from <https://code.google.com/archive/p/word2vec/>.

9376 preference for coherence motivates **collectively** linking these references to the first three
 9377 U.S. presidents.

9378 A general approach to collective entity linking is to introduce a compatibility score
 9379 $\psi_c(\mathbf{y})$. Collective entity linking is then performed by optimizing the global objective,

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathbb{Y}(\mathbf{x})}{\operatorname{argmax}} \Psi_c(\mathbf{y}) + \sum_{i=1}^N \Psi_\ell(y^{(i)}, \mathbf{x}^{(i)}, \mathbf{c}^{(i)}), \quad [17.5]$$

9380 where $\mathbb{Y}(\mathbf{x})$ is the set of all possible collective entity assignments for the mentions in \mathbf{x} ,
 9381 and ψ_ℓ is the local scoring function for each entity i . The compatibility function is typically
 9382 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
 9383 can be computed in a number of different ways:

- 9384 • Wikipedia defines high-level categories for entities (e.g., *living people*, *Presidents of*
 9385 *the United States*, *States of the United States*), and Ψ_c can reward entity pairs for the
 9386 number of categories that they have in common (Cucerzan, 2007).
- 9387 • Compatibility can be measured by the number of incoming hyperlinks shared by
 9388 the Wikipedia pages for the two entities (Milne and Witten, 2008).
- 9389 • In a neural architecture, the compatibility of two entities can be set equal to the inner
 9390 product of their embeddings, $\Psi_c(y^{(i)}, y^{(j)}) = \mathbf{v}_{y^{(i)}} \cdot \mathbf{v}_{y^{(j)}}$.
- 9391 • A non-pairwise compatibility score can be defined using a type of latent variable
 9392 model known as a **probabilistic topic model** (Blei et al., 2003; Blei, 2012). In this
 9393 framework, each latent topic is a probability distribution over entities, and each
 9394 document has a probability distribution over topics. Each entity helps to determine
 9395 the document's distribution over topics, and in turn these topics help to resolve am-
 9396 biguous entity mentions (Newman et al., 2006). Inference can be performed using
 9397 the sampling techniques described in chapter 5.

9398 Unfortunately, collective entity linking is **NP-hard** even for pairwise compatibility func-
 9399 tions, so exact optimization is almost certainly intractable. Various approximate inference
 9400 techniques have been proposed, including **integer linear programming** (Cheng and Roth,
 9401 2013), **Gibbs sampling** (Han and Sun, 2012), and graph-based algorithms (Hoffart et al.,
 9402 2011; Han et al., 2011).

9403 17.1.3 *Pairwise ranking loss functions

9404 The loss function defined in Equation 17.2 considers only the highest-scoring prediction
 9405 \hat{y} , but in fact, the true entity $y^{(i)}$ should outscore *all* other entities. A loss function based on
 9406 this idea would give a gradient against the features or representations of several entities,

Algorithm 18 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

```

9407 not just the top-scoring prediction. Usunier et al. (2009) define a general ranking error
 9408 function,

$$L_{\text{rank}}(k) = \sum_{j=1}^k \alpha_j, \quad \text{with } \alpha_1 \geq \alpha_2 \geq \dots \geq 0, \quad [17.6]$$

9409 where k is equal to the number of labels ranked higher than the correct label $y^{(i)}$. This
 9410 function defines a class of ranking errors: if $\alpha_j = 1$ for all j , then the ranking error is
 9411 equal to the rank of the correct entity; if $\alpha_1 = 1$ and $\alpha_{j>1} = 0$, then the ranking error is
 9412 one whenever the correct entity is not ranked first; if α_j decreases smoothly with j , as in
 9413 $\alpha_j = \frac{1}{j}$, 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,

$$r(y^{(i)}, \mathbf{x}^{(i)}) \triangleq \sum_{y \in \mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}} \delta \left(1 + \psi(y, \mathbf{x}^{(i)}) \geq \psi(y^{(i)}, \mathbf{x}^{(i)}) \right), \quad [17.7]$$

9414 where $\delta(\cdot)$ is a delta function, and $\mathcal{Y}(\mathbf{x}^{(i)}) \setminus y^{(i)}$ is the set of all entity candidates minus
 9415 the true entity $y^{(i)}$. The margin-augmented rank is the rank of the true entity, after aug-
 9416 menting every other candidate with a margin of one, under the current scoring function
 9417 ψ . (The context c is omitted for clarity, and can be considered part of x .)

For each instance, a hinge loss is computed from the ranking error associated with this

margin-augmented rank, and the violation of the margin constraint,

$$\ell(y^{(i)}, \mathbf{x}^{(i)}) = \frac{L_{\text{rank}}(r(y^{(i)}, \mathbf{x}^{(i)}))}{r(y^{(i)}, \mathbf{x}^{(i)})} \sum_{y \in \mathcal{Y}(\mathbf{x}) \setminus y^{(i)}} \left(\psi(y, \mathbf{x}^{(i)}) - \psi(y^{(i)}, \mathbf{x}^{(i)}) + 1 \right)_+, \quad [17.8]$$

The sum in Equation 17.8 includes non-zero values for every label that is ranked at least as high as the true entity, after applying the margin augmentation. Dividing by the margin-augmented rank of the true entity thus gives the average violation.

The objective in Equation 17.8 is expensive to optimize when the label space is large, as is usually the case for entity linking against large knowledge bases. This motivates a randomized approximation called **WARP** (Weston et al., 2011), shown in Algorithm 18. In this procedure, we sample random entities until one violates the pairwise margin constraint, $\psi(y, \mathbf{x}^{(i)}) + 1 \geq \psi(y^{(i)}, \mathbf{x}^{(i)})$. The number of samples N required to find such a violation yields an approximation of the margin-augmented rank of the true entity, $r(y^{(i)}, \mathbf{x}^{(i)}) \approx \left\lfloor \frac{|\mathcal{Y}(\mathbf{x})|}{N} \right\rfloor$. If a violation is found immediately, $N = 1$, the correct entity probably ranks below many others, $r \approx |\mathcal{Y}(\mathbf{x})|$. If many samples are required before a violation is found, $N \rightarrow |\mathcal{Y}(\mathbf{x})|$, then the correct entity is probably highly ranked, $r \rightarrow 1$. A computational advantage of WARP is that it is not necessary to find the highest-scoring label, which can impose a non-trivial computational cost when $\mathcal{Y}(\mathbf{x}^{(i)})$ is large. The objective is conceptually similar to the **negative sampling** objective in WORD2VEC (chapter 14), which compares the observed word against randomly sampled alternatives.

17.2 Relations

After identifying the entities that are mentioned in a text, the next step is to determine how they are related. Consider the following example:

(17.8) George Bush traveled to France on Thursday for a summit.

This sentence introduces a relation between the entities referenced by *George Bush* and *France*. In the Automatic Content Extraction (ACE) ontology (Linguistic Data Consortium, 2005), the type of this relation is PHYSICAL, and the subtype is LOCATED. This relation would be written,

$$\text{PHYSICAL.LOCATED(GEORGE BUSH, FRANCE).} \quad [17.9]$$

Relations take exactly two arguments, and the order of the arguments matters.

In the ACE datasets, relations are annotated between entity mentions, as in the example above. Relations can also hold between nominals, as in the following example from the SemEval-2010 shared task (Hendrickx et al., 2009):

| | |
|---------------------|---|
| CAUSE-EFFECT | <i>those cancers were caused by radiation exposures</i> |
| INSTRUMENT-AGENCY | <i>phone operator</i> |
| PRODUCT-PRODUCER | <i>a factory manufactures suits</i> |
| CONTENT-CONTAINER | <i>a bottle of honey was weighed</i> |
| ENTITY-ORIGIN | <i>letters from foreign countries</i> |
| ENTITY-DESTINATION | <i>the boy went to bed</i> |
| COMPONENT-WHOLE | <i>my apartment has a large kitchen</i> |
| MEMBER-COLLECTION | <i>there are many trees in the forest</i> |
| COMMUNICATION-TOPIC | <i>the lecture was about semantics</i> |

Table 17.1: Relations and example sentences from the SemEval-2010 dataset (Hendrickx et al., 2009)

9446 (17.9) The cup contained tea from dried ginseng.

9447 This sentence describes a relation of type ENTITY-ORIGIN between *tea* and *ginseng*. Nominal
 9448 relation extraction is closely related to **semantic role labeling** (chapter 13). The main
 9449 difference is that relation extraction is restricted to a relatively small number of relation
 9450 types; for example, Table 17.1 shows the ten relation types from SemEval-2010.

9451 17.2.1 Pattern-based relation extraction

9452 Early work on relation extraction focused on hand-crafted patterns (Hearst, 1992). For
 9453 example, the appositive *Starbuck, a native of Nantucket* signals the relation ENTITY-ORIGIN
 9454 between *Starbuck* and *Nantucket*. This pattern can be written as,

$$\text{PERSON , } a \text{ native of LOCATION} \Rightarrow \text{ENTITY-ORIGIN(PERSON, LOCATION)}. \quad [17.10]$$

9455 This pattern will be “triggered” whenever the literal string *, a native of* occurs between an
 9456 entity of type PERSON and an entity of type LOCATION. Such patterns can be generalized
 9457 beyond literal matches using techniques such as lemmatization, which would enable the
 9458 words (*buy, buys, buying*) to trigger the same patterns (see § 4.3.1). A more aggressive
 9459 strategy would be to group all words in a WordNet synset (§ 4.2), so that, e.g., *buy* and
 9460 *purchase* trigger the same patterns.

9461 Relation extraction patterns can be implemented in finite-state automata (§ 9.1). If the
 9462 named entity recognizer is also a finite-state machine, then the systems can be combined
 9463 by finite-state transduction (Hobbs et al., 1997). This makes it possible to propagate uncer-
 9464 tainty through the finite-state cascade, and disambiguate from higher-level context. For
 9465 example, suppose the entity recognizer cannot decide whether *Starbuck* refers to either a
 9466 PERSON or a LOCATION; in the composed transducer, the relation extractor would be free
 9467 to select the PERSON annotation when it appears in the context of an appropriate pattern.

9468 **17.2.2 Relation extraction as a classification task**

9469 Relation extraction can be formulated as a classification problem,

$$\hat{r}_{(i,j),(m,n)} = \operatorname{argmax}_{r \in \mathcal{R}} \Psi(r, (i, j), (m, n), \mathbf{w}), \quad [17.11]$$

9470 where $r \in \mathcal{R}$ is a relation type (possibly NIL), $\mathbf{w}_{i+1:j}$ is the span of the first argument, and
 9471 $\mathbf{w}_{m+1:n}$ is the span of the second argument. The argument $\mathbf{w}_{m+1:n}$ may appear before
 9472 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
 9473 argument of the relation. We now consider three alternatives for computing the scoring
 9474 function.

9475 **Feature-based classification**

9476 In a feature-based classifier, the scoring function is defined as,

$$\Psi(r, (i, j), (m, n), \mathbf{w}) = \boldsymbol{\theta} \cdot \mathbf{f}(r, (i, j), (m, n), \mathbf{w}), \quad [17.12]$$

9477 with $\boldsymbol{\theta}$ representing a vector of weights, and $\mathbf{f}(\cdot)$ a vector of features. The pattern-based
 9478 methods described in § 17.2.1 suggest several features:

- 9479 • Local features of $\mathbf{w}_{i+1:j}$ and $\mathbf{w}_{m+1:n}$, including: the strings themselves; whether they
 9480 are recognized as entities, and if so, which type; whether the strings are present in a
 9481 **gazetteer** of entity names; each string's syntactic head (§ 9.2.2).
- 9482 • Features of the span between the two arguments, $\mathbf{w}_{j+1:m}$ or $\mathbf{w}_{n+1:i}$ (depending on
 9483 which argument appears first): the length of the span; the specific words that appear
 9484 in the span, either as a literal sequence or a bag-of-words; the wordnet synsets (§ 4.2)
 9485 that appear in the span between the arguments.
- 9486 • Features of the syntactic relationship between the two arguments, typically the **de-**
 9487 **pendency path** between the arguments (§ 13.2.1). Example dependency paths are
 9488 shown in Table 17.2.

9489 **Kernels**

9490 Suppose that the first line of Table 17.2 is a labeled example, and the remaining lines are
 9491 instances to be classified. A feature-based approach would have to decompose the depen-
 9492 dency paths into features that capture individual edges, with or without their labels, and
 9493 then learn weights for each of these features: for example, the second line contains identi-
 9494 cal dependencies, but different arguments; the third line contains a different inflection of
 9495 the word *travel*; the fourth and fifth lines each contain an additional edge on the depen-
 9496 dency path; and the sixth example uses an entirely different path. Rather than attempting
 9497 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

9498 are similar and different, we can instead define a similarity function κ , which computes a
9499 score for any pair of instances, $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}_+$. The score for any pair of instances (i, j)
9500 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
9501 function κ obeys a few key properties it is a valid **kernel function**.⁴

Given a valid kernel function, we can build a non-linear classifier without explicitly defining a feature vector or neural network architecture. For a binary classification problem $y \in \{-1, 1\}$, we have the decision function,

$$\hat{y} = \text{Sign}(b + \sum_{i=1}^N y^{(i)} \alpha^{(i)} \kappa(\mathbf{x}^{(i)}, \mathbf{x})) \quad [17.13]$$

9502 where b and $\{\alpha^{(i)}\}_{i=1}^N$ are parameters that must be learned from the training set, under
9503 the constraint $\forall_i, \alpha^{(i)} \geq 0$. Intuitively, each α_i specifies the importance of the instance $\mathbf{x}^{(i)}$
9504 towards the classification rule. Kernel-based classification can be viewed as a weighted
9505 form of the **nearest-neighbor** classifier (Hastie et al., 2009), in which test instances are
9506 assigned the most common label among their near neighbors in the training set. This
9507 results in a non-linear classification boundary. The parameters are typically learned from
9508 a margin-based objective (see § 2.4), leading to the **kernel support vector machine**. To
9509 generalize to multi-class classification, we can train separate binary classifiers for each
9510 label (sometimes called **one-versus-all**), or train binary classifiers for each pair of possible
9511 labels (**one-versus-one**).

9512 Dependency kernels are particularly effective for relation extraction, due to their ability
9513 to capture syntactic properties of the path between the two candidate arguments. One
9514 class of dependency tree kernels is defined recursively, with the score for a pair of trees

⁴The **Gram matrix** \mathbf{K} arises from computing the kernel function between all pairs in a set of instances. For a valid kernel, the Gram matrix must be symmetric ($\mathbf{K} = \mathbf{K}^\top$) and positive semi-definite ($\forall \mathbf{a}, \mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0$). For more on kernel-based classification, see chapter 14 of Murphy (2012).

equal to the similarity of the root nodes and the sum of similarities of matched pairs of child subtrees (Zelenko et al., 2003; Culotta and Sorensen, 2004). Alternatively, Bunescu and Mooney (2005) define a kernel function over sequences of unlabeled dependency edges, in which the score is computed as a product of scores for each pair of words in the sequence: identical words receive a high score, words that share a synset or part-of-speech receive a small non-zero score (e.g., *travel* / *visit*), and unrelated words receive a score of zero.

Neural relation extraction

Convolutional neural networks (§ 3.4) 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 x_m , and two positional embeddings, specifying the position with respect to a_1 and a_2 . The matrix $\mathbf{X}(a_1, a_2)$ is then taken as input to a convolutional layer (see § 3.4), and max-pooling is applied to obtain a vector. The final scoring function is then,

$$\Psi(r, i, j, \mathbf{X}) = \theta_r \cdot \text{MaxPool}(\text{ConvNet}(\mathbf{X}(i, j); \phi)), \quad [17.15]$$

where ϕ defines the parameters of the convolutional operator, and the θ_r defines a set of weights for relation r . The model can be trained using a margin objective,

$$\hat{r} = \underset{r}{\operatorname{argmax}} \Psi(r, i, j, \mathbf{X}) \quad [17.16]$$

$$\ell = (1 + \psi(\hat{r}, i, j, \mathbf{X}) - \psi(r, i, j, \mathbf{X}))_+. \quad [17.17]$$

Recurrent neural networks (§ 6.3) have also been applied to relation extraction, using a network such as a 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 Bush* $\xleftarrow{\text{NSUBJ}}$ *wants* $\xrightarrow{\text{XCOMP}}$ *travel* \rightarrow_{OBL} *France* is segmented into the

subpaths, $George \xleftarrow{\text{NSUBJ}} Bush \xleftarrow{\text{wants}} \text{and} \xleftarrow{\text{wants}} \text{wants} \xrightarrow{\text{XCOMP}} travel \xrightarrow{\text{OBL}} France$. In each path, a recurrent neural network is run from the argument to the root word (in this case, *wants*). The final representation by max pooling (§ 3.4) 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 (e.g., *France-nation*; see § 4.2). To define the model formally, let $s(m)$ define the successor of word m in either the left or right subpath (in a dependency path, each word can have a successor in at most one subpath). Let $\mathbf{x}_m^{(c)}$ indicate the embedding of word (or relation) m in channel c , and let $\overleftarrow{\mathbf{h}}_m^{(c)}$ and $\overrightarrow{\mathbf{h}}_m^{(c)}$ indicate the associated recurrent states in the left and right subtrees respectively. Then the complete model is specified as follows,

$$\mathbf{h}_{s(m)}^{(c)} = \text{RNN}(\mathbf{x}_{s(m)}^{(c)}, \mathbf{h}_m^{(c)}) \quad [17.18]$$

$$\mathbf{z}^{(c)} = \text{MaxPool}(\overleftarrow{\mathbf{h}}_i^{(c)}, \overleftarrow{\mathbf{h}}_{s(i)}^{(c)}, \dots, \overleftarrow{\mathbf{h}}_{\text{root}}^{(c)}, \overrightarrow{\mathbf{h}}_j^{(c)}, \overrightarrow{\mathbf{h}}_{s(j)}^{(c)}, \dots, \overrightarrow{\mathbf{h}}_{\text{root}}^{(c)}) \quad [17.19]$$

$$\Psi(r, i, j) = \boldsymbol{\theta} \cdot [\mathbf{z}^{(\text{word})}; \mathbf{z}^{(\text{POS})}; \mathbf{z}^{(\text{dependency})}; \mathbf{z}^{(\text{hypernym})}] \quad [17.20]$$

9536 Note that \mathbf{z} is computed by applying max-pooling to the *matrix* of horizontally concatenated
 9537 vectors \mathbf{h} , while Ψ is computed from the *vector* of vertically concatenated vectors
 9538 \mathbf{z} . Xu et al. (2015) pass the score Ψ through a **softmax** layer to obtain a probability
 9539 $p(r | i, j, \mathbf{w})$, and train the model by regularized **cross-entropy**. Miwa and Bansal (2016)
 9540 show that a related model can solve the more challenging “end-to-end” relation extrac-
 9541 tion task, in which the model must simultaneously detect entities and then extract their
 9542 relations.

9543 17.2.3 Knowledge base population

9544 In many applications, what matters is not what fraction of sentences are analyzed cor-
 9545 rectly, but how much accurate knowledge can be extracted. **Knowledge base population**
 9546 (**KBP**) refers to the task of filling in Wikipedia-style infoboxes, as shown in Figure 17.1a.
 9547 Knowledge base population can be decomposed into two subtasks: **entity linking** (de-
 9548 scribed in § 17.1), and **slot filling** (Ji and Grishman, 2011). Slot filling has two key dif-
 9549 ferences from the formulation of relation extraction presented above: the relations hold
 9550 between entities rather than spans of text, and the performance is evaluated at the *type*
 9551 *level* (on entity pairs), rather than on the *token level* (on individual sentences).

9552 From a practical standpoint, there are three other important differences between slot
 9553 filling and per-sentence relation extraction.

- 9554 • KBP tasks are often formulated from the perspective of identifying attributes of a
 9555 few “query” entities. As a result, these systems often start with an **information**
 9556 **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.

Information fusion

In knowledge base population, there will often be multiple pieces of evidence for (and sometimes against) a single relation. For example, a search for the entity MAYNARD JACKSON, JR. may return several passages that reference the entity ATLANTA:⁵

- (17.10)
- a. Elected mayor of **Atlanta** in 1973, **Maynard Jackson** was the first African American to serve as mayor of a major southern city.
 - b. **Atlanta**'s airport will be renamed to honor **Maynard Jackson**, the city's first Black mayor.
 - c. Born in Dallas, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to **Atlanta** when he was 8.
 - d. **Maynard Jackson** has gone from one of the worst high schools in **Atlanta** to one of the best.

The first and second examples provide evidence for the relation MAYOR holding between the entities ATLANTA and MAYNARD JACKSON, JR.. The third example provides evidence for a different relation between these same entities, LIVED-IN. The fourth example poses an entity linking problem, referring to MAYNARD JACKSON HIGH SCHOOL. Knowledge base population requires aggregating this sort of textual evidence, and predicting the relations that are most likely to hold.

One approach is to run a single-document relation extraction system (using the techniques described in § 17.2.2), and then aggregate the results (Li et al., 2011). Relations that are detected with high confidence in multiple documents are more likely to be valid,

⁵First three examples from: <http://www.georgiaencyclopedia.org/articles/government-politics/maynard-jackson-1938-2003>; JET magazine, November 10, 2003; www.todayingeorgiahistory.org/content/maynard-jackson-elected

9588 motivating the heuristic,

$$\psi(r, e_1, e_2) = \sum_{i=1}^N (\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)}))^{\alpha}, \quad [17.21]$$

9589 where $\text{p}(r(e_1, e_2) | \mathbf{w}^{(i)})$ is the probability of relation r between entities e_1 and e_2 conditioned
 9590 on the text $\mathbf{w}^{(i)}$, and $\alpha \gg 1$ is a tunable hyperparameter. Using this heuristic, it is
 9591 possible to rank all candidate relations, and trace out a **precision-recall curve** as more re-
 9592 lations are extracted.⁶ Alternatively, features can be aggregated across multiple passages
 9593 of text, feeding a single type-level relation extraction system (Wolfe et al., 2017).

9594 Precision can be improved by introducing constraints across multiple relations. For
 9595 example, if we are certain of the relation $\text{PARENT}(e_1, e_2)$, then it cannot also be the case
 9596 that $\text{PARENT}(e_2, e_1)$. Integer linear programming makes it possible to incorporate such
 9597 constraints into a global optimization (Li et al., 2011). Other pairs of relations have pos-
 9598 itive correlations, such $\text{MAYOR}(e_1, e_2)$ and $\text{LIVED-IN}(e_1, e_2)$. Compatibility across relation
 9599 types can be incorporated into probabilistic graphical models (e.g., Riedel et al., 2010).

9600 Distant supervision

9601 Relation extraction is “annotation hungry,” because each relation requires its own la-
 9602 beled data. Rather than relying on annotations of individual documents, it would be
 9603 preferable to use existing knowledge resources — such as the many facts that are al-
 9604 ready captured in knowledge bases like DBpedia. However such annotations raise the
 9605 inverse of the information fusion problem considered above: the existence of the relation
 9606 $\text{MAYOR}(\text{MAYNARD JACKSON JR., ATLANTA})$ provides only **distant supervision** for the
 9607 example texts in which this entity pair is mentioned.

9608 One approach is to treat the entity pair as the instance, rather than the text itself (Mintz
 9609 et al., 2009). Features are then aggregated across all sentences in which both entities are
 9610 mentioned, and labels correspond to the relation (if any) between the entities in a knowl-
 9611 edge base, such as FreeBase. Negative instances are constructed from entity pairs that are
 9612 not related in the knowledge base. In some cases, two entities are related, but the knowl-
 9613 edge base is missing the relation; however, because the number of possible entity pairs is
 9614 huge, these missing relations are presumed to be relatively rare. This approach is shown
 9615 in Figure 17.2.

9616 In **multiple instance learning**, labels are assigned to *sets* of instances, of which only
 9617 an unknown subset are actually relevant (Dietterich et al., 1997; Maron and Lozano-Pérez,
 9618 1998). This formalizes the framework of distant supervision: the relation $\text{REL}(A, B)$ acts
 9619 as a label for the entire set of sentences mentioning entities A and B, even when only a

⁶The precision-recall curve is similar to the ROC curve shown in Figure 4.4, but it includes the precision $\frac{\text{TP}}{\text{TP} + \text{FP}}$ rather than the false positive rate $\frac{\text{FP}}{\text{FP} + \text{TN}}$.

- **Label** : MAYOR(ATLANTA, MAYNARD JACKSON)
 - Elected mayor of **Atlanta** in 1973, **Maynard Jackson** ...
 - **Atlanta**'s airport will be renamed to honor **Maynard Jackson**, the city's first Black mayor
 - Born in Dallas, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to **Atlanta** when he was 8.
- **Label** : MAYOR(NEW YORK, FIORELLO LA GUARDIA)
 - **Fiorello La Guardia** was Mayor of **New York** for three terms ...
 - **Fiorello La Guardia**, then serving on the **New York** City Board of Aldermen...
- **Label** : BORN-IN(DALLAS, MAYNARD JACKSON)
 - Born in **Dallas**, Texas in 1938, **Maynard Holbrook Jackson, Jr.** moved to Atlanta when he was 8.
 - **Maynard Jackson** was raised in **Dallas** ...
- **Label** : NIL(NEW YORK, MAYNARD JACKSON)
 - **Jackson** married Valerie Richardson, whom he had met in **New York**...
 - **Jackson** was a member of the Georgia and **New York** bars ...

Figure 17.2: Four training instances for relation classification using **distant supervision** Mintz et al. (2009). The first two instances are positive for the MAYOR relation, and the third instance is positive for the BORN-IN relation. The fourth instance is a negative example, constructed from a pair of entities (NEW YORK, MAYNARD JACKSON) that do not appear in any Freebase relation. Each instance's features are computed by aggregating across all sentences in which the two entities are mentioned.

9620 subset of these sentences actually describes the relation. One approach to multi-instance
 9621 learning is to introduce a binary **latent variable** for each sentence, indicating whether the
 9622 sentence expresses the labeled relation (Riedel et al., 2010). A variety of inference tech-
 9623 niques have been employed for this probabilistic model of relation extraction: Surdeanu
 9624 et al. (2012) use expectation maximization, Riedel et al. (2010) use sampling, and Hoff-
 9625 mann et al. (2011) use a custom graph-based algorithm. Expectation maximization and
 9626 sampling are surveyed in chapter 5, and are covered in more detail by Murphy (2012);
 9627 graph-based methods are surveyed by Mihalcea and Radev (2011).

9628 17.2.4 Open information extraction

9629 In classical relation extraction, the set of relations is defined in advance, using a **schema**.
 9630 The relation for any pair of entities can then be predicted using multi-class classification.
 9631 In **open information extraction** (OpenIE), a relation can be any triple of text. The example
 9632 sentence (17.10a) instantiates several “relations” of this sort, e.g.,

| 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.

- 9633 • (*mayor of, Maynard Jackson, Atlanta*),
- 9634 • (*elected, Maynard Jackson, mayor of Atlanta*),
- 9635 • (*elected in, Maynard Jackson, 1973*).

9636 Extracting such tuples can be viewed as a lightweight version of **semantic role labeling**
 9637 (chapter 13), with only two argument types: first slot and second slot. The task is gen-
 9638 erally evaluated on the relation level, rather than on the level of sentences: precision is
 9639 measured by the number of extracted relations that are accurate, and recall is measured
 9640 by the number of true relations that were successfully extracted. OpenIE systems are
 9641 trained from distant supervision or bootstrapping, rather than from labeled sentences.

9642 An early example is the TEXTRUNNER system (Banko et al., 2007), which identifies
 9643 relations with a set of handcrafted syntactic rules. The examples that are acquired from
 9644 the handcrafted rules are then used to train a classification model that uses part-of-speech
 9645 patterns as features. Finally, the relations that are extracted by the classifier are aggre-
 9646 gated, removing redundant relations and computing the number of times that each rela-
 9647 tion is mentioned in the corpus. TEXTRUNNER was the first in a series of systems that
 9648 performed increasingly accurate open relation extraction by incorporating more precise
 9649 linguistic features (Etzioni et al., 2011), distant supervision from Wikipedia infoboxes (Wu
 9650 and Weld, 2010), and better learning algorithms (Zhu et al., 2009).

9651 17.3 Events

9652 Relations link pairs of entities, but many real-world situations involve more than two en-
 9653 tities. Consider again the example sentence (17.10a), which describes the event of an elec-
 9654 tion, with four properties: the office (MAYOR), the district (ATLANTA), the date (1973), and
 9655 the person elected (MAYNARD JACKSON, JR.). In **event detection**, a schema is provided

for each event type (e.g., an election, a terrorist attack, or a chemical reaction), indicating all the possible properties of the event. The system is then required to fill in as many of these properties as possible (Doddington et al., 2004).

Event detection systems generally involve a retrieval component (finding relevant documents and passages of text) and an extraction component (determining the properties of the event based on the retrieved texts). Early approaches focused on finite-state patterns for identify event properties (Hobbs et al., 1997); such patterns can be automatically induced by searching for patterns that are especially likely to appear in documents that match the event query (Riloff, 1996). Contemporary approaches employ techniques that are similar to FrameNet semantic role labeling (§ 13.2), such as structured prediction over local and global features (Li et al., 2013) and bidirectional recurrent neural networks (Feng et al., 2016). These methods detect whether an event is described in a sentence, and if so, what are its properties.

Event coreference Because multiple sentences may describe unique properties of a single event, **event coreference** is required to link event mentions across a single passage of text, or between passages (Humphreys et al., 1997). Bejan and Harabagiu (2014) define event coreference as the task of identifying event mentions that share the same event participants (i.e., the slot-filling entities) and the same event properties (e.g., the time and location), within or across documents. Event coreference resolution can be performed using supervised learning techniques in a similar way to entity coreference, as described in chapter 15: move left-to-right through the document, and use a classifier to decide whether to link each event reference to an existing cluster of coreferent events, or to create a new cluster (Ahn, 2006). Each clustering decision is based on the compatibility of features describing the participants and properties of the event. Due to the difficulty of annotating large amounts of data for entity coreference, unsupervised approaches are especially desirable (Chen and Ji, 2009; Bejan and Harabagiu, 2014).

Relations between events Just as entities are related to other entities, events may be related to other events: for example, the event of winning an election both *precedes* and *causes* the event of serving as mayor; moving to Atlanta *precedes* and *enables* the event of becoming mayor of Atlanta; moving from Dallas to Atlanta *prevents* the event of later becoming mayor of Dallas. As these examples show, events may be related both temporally and causally. The **TimeML** annotation scheme specifies a set of six temporal relations between events (Pustejovsky et al., 2005), derived in part from **interval algebra** (Allen, 1984). The TimeBank corpus provides TimeML annotations for 186 documents (Pustejovsky et al., 2003). Methods for detecting these temporal relations combine supervised machine learning with temporal constraints, such as transitivity (e.g. Mani et al., 2006; Chambers and Jurafsky, 2008).

More recent annotation schemes and datasets combine temporal and causal relations (Mirza

| | 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.

et al., 2014; Dunietz et al., 2017): for example, the CaTeRS dataset includes annotations of 320 five-sentence short stories (Mostafazadeh et al., 2016). Abstracting still further, **processes** are networks of causal relations between multiple events. A small dataset of biological processes is annotated in the ProcessBank dataset (Berant et al., 2014), with the goal of supporting automatic question answering on scientific textbooks.

17.4 Hedges, denials, and hypotheticals

The methods described thus far apply to **propositions** about the way things are in the real world. But natural language can also describe events and relations that are likely or unlikely, possible or impossible, desired or feared. The following examples hint at the scope of the problem (Prabhakaran et al., 2010):

- 9704 (17.11) a. GM will lay off workers.
- 9705 b. A spokesman for GM said GM will lay off workers.
- 9706 c. GM may lay off workers.
- 9707 d. The politician claimed that GM will lay off workers.
- 9708 e. Some wish GM would lay off workers.
- 9709 f. Will GM lay off workers?
- 9710 g. Many wonder whether GM will lay off workers.

Accurate information extraction requires handling these **extra-propositional** aspects of meaning, which are sometimes summarized under the terms **modality** and **negation**.⁷

⁷The classification of negation as extra-propositional is controversial: Packard et al. (2014) argue that negation is a “core part of compositionally constructed logical-form representations.” Negation is an element of the semantic parsing tasks discussed in chapter 12 and chapter 13 — for example, negation markers are treated as adjuncts in PropBank semantic role labeling. However, many of the relation extraction methods mentioned in this chapter do not handle negation directly. A further consideration is that negation interacts closely with aspects of modality that are generally not considered in propositional semantics, such as certainty and subjectivity.

9713 Modality refers to expressions of the speaker's attitude towards her own statements, in-
9714 cluding "degree of certainty, reliability, subjectivity, sources of information, and perspec-
9715 tive" (Morante and Sporleder, 2012). Various systematizations of modality have been
9716 proposed (e.g., Palmer, 2001), including categories such as future, interrogative, imper-
9717 ative, conditional, and subjective. Information extraction is particularly concerned with
9718 negation and certainty. For example, Saurí and Pustejovsky (2009) link negation with
9719 a modal calculus of certainty, likelihood, and possibility, creating the two-dimensional
9720 schema shown in Table 17.4. This is the basis for the FactBank corpus, with annotations
9721 of the **factuality** of all sentences in 208 documents of news text.

9722 A related concept is **hedging**, in which speakers limit their commitment to a proposi-
9723 tion (Lakoff, 1973):

9724 (17.12) These results **suggest** that expression of c-jun, jun B and jun D genes **might** be in-
9725 volved in terminal granulocyte differentiation... (Morante and Daelemans, 2009)

9726 (17.13) A whale is **technically** a mammal (Lakoff, 1973)

9727 In the first example, the hedges *suggest* and *might* communicate uncertainty; in the second
9728 example, there is no uncertainty, but the hedge *technically* indicates that the evidence for
9729 the proposition will not fully meet the reader's expectations. Hedging has been studied
9730 extensively in scientific texts (Medlock and Briscoe, 2007; Morante and Daelemans, 2009),
9731 where the goal of large-scale extraction of scientific facts is obstructed by hedges and spec-
9732 ulation. Still another related aspect of modality is **evidentiality**, in which speakers mark
9733 the source of their information. In many languages, it is obligatory to mark evidentiality
9734 through affixes or particles (Aikhenvald, 2004); while evidentiality is not grammaticalized
9735 in English, authors are expected to express this information in contexts such as journal-
9736 ism (Kovach and Rosenstiel, 2014) and Wikipedia.⁸

9737 Methods for handling negation and modality generally include two phases:

- 9738 1. detecting negated or uncertain events;
9739 2. identifying **scope** of the negation or modal operator.

9740 A considerable body of work on negation has employed rule-based techniques such
9741 as regular expressions (Chapman et al., 2001) to detect negated events. Such techniques
9742 match lexical cues (e.g., *Norwood was **not** elected Mayor*), while avoiding "double nega-
9743 tives" (e.g., *surely all this is **not without** meaning*). Supervised techniques involve classi-
9744 fiers over lexical and syntactic features (Uzuner et al., 2009) and sequence labeling (Prab-
9745 hakaran et al., 2010).

⁸<https://en.wikipedia.org/wiki/Wikipedia:Verifiability>

9746 The scope refers to the elements of the text whose propositional meaning is negated or
 9747 modulated (Huddleston and Pullum, 2005), as elucidated in the following example from
 9748 Morante and Sporleder (2012):

- 9749 (17.14) [After his habit he said] **nothing**, and after mine I asked no questions.
 9750 After his habit he said nothing, and [after mine I asked] **no** [questions].

9751 In this sentence, there are two negation cues (*nothing* and *no*). Each negates an event, in-
 9752 dicated by the underlined verbs *said* and *asked*, and each occurs within a scope: *after his*
 9753 *habit he said* and *after mine I asked* *questions*. Scope identification is typically formal-
 9754 ized as sequence labeling problems, with each word token labeled as beginning, inside,
 9755 or outside of a cue, focus, or scope span (see § 8.3). Conventional sequence labeling ap-
 9756 proaches can then be applied, using surface features as well as syntax (Veldal et al., 2012)
 9757 and semantic analysis (Packard et al., 2014). Labeled datasets include the BioScope corpus
 9758 of biomedical texts (Vincze et al., 2008) and a shared task dataset of detective stories by
 9759 Arthur Conan Doyle (Morante and Blanco, 2012).

9760 17.5 Question answering and machine reading

9761 The victory of the Watson question-answering system against three top human players on
 9762 the game show *Jeopardy!* was a landmark moment for natural language processing (Fer-
 9763 rucci et al., 2010). Game show questions are usually answered by **factoids**: entity names
 9764 and short phrases.⁹ The task of factoid question answering is therefore closely related to
 9765 information extraction, with the additional problem of accurately parsing the question.

9766 17.5.1 Formal semantics

9767 Semantic parsing is an effective method for question-answering in restricted domains
 9768 such as questions about geography and airline reservations (Zettlemoyer and Collins,
 9769 2005), and has also been applied in “open-domain” settings such as question answering
 9770 on Freebase (Berant et al., 2013) and biomedical research abstracts (Poon and Domingos,
 9771 2009). One approach is to convert the question into a lambda calculus expression that
 9772 returns a boolean value: for example, the question *who is the mayor of the capital of Georgia?*
 9773 would be converted to,

$$\lambda x. \exists y \text{ CAPITAL(GEORGIA, } y) \wedge \text{MAYOR}(y, x). \quad [17.22]$$

9774 This lambda expression can then be used to query an existing knowledge base, returning
 9775 “true” for all entities that satisfy it.

⁹The broader landscape of question answering includes “why” questions (*Why did Ahab continue to pursue the white whale?*), “how questions” (*How did Queequeg die?*), and requests for summaries (*What was Ishmael’s attitude towards organized religion?*). For more, see Hirschman and Gaizauskas (2001).

17.5.2 Machine reading

Recent work has focused on answering questions about specific textual passages, similar to the reading comprehension examinations for young students (Hirschman et al., 1999). This task has come to be known as **machine reading**.

Datasets

The machine reading problem can be formulated in a number of different ways. The most important distinction is what form the answer should take.

- **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, the answer is deducible from the text alone, while in the science exams, the system must make inferences using an existing model of the underlying scientific phenomena. Here is an example from MCTest:

(17.15) James the turtle was always getting into trouble. Sometimes he'd reach into the freezer and empty out all the food ...

- Q: What is the name of the trouble making turtle?
(a) Fries
(b) Pudding
(c) James
(d) Jane

- **Cloze-style “fill in the blank” questions**, as in the CNN/Daily Mail comprehension task (Hermann et al., 2015), the Children’s Book Test (Hill et al., 2016), and the Who-did-What dataset (Onishi et al., 2016). In these tasks, the system must guess which word or entity completes a sentence, based on reading a passage of text. Here is an example from Who-did-What:

(17.16) Q: Tottenham manager Juande Ramos has hinted he will allow _____ to leave if the Bulgaria striker makes it clear he is unhappy. (Onishi et al., 2016)

The query sentence may be selected either from the story itself, or from an external summary. In either case, datasets can be created automatically by processing large quantities existing documents. An additional constraint is that that missing element from the cloze must appear in the main passage of text: for example, in Who-did-What, the candidates include all entities mentioned in the main passage. In the CNN/Daily Mail dataset, each entity name is replaced by a unique identifier, e.g., ENTITY37. This ensures that correct answers can only be obtained by accurately reading the text, and not from external knowledge about the entities.

- 9810 • **Extractive** question answering, in which the answer is drawn from the original text.
 9811 In WikiQA, answers are sentences (Yang et al., 2015). In the Stanford Question An-
 9812 swering Dataset (SQuAD), answers are words or short phrases (Rajpurkar et al.,
 9813 2016):

9814 (17.17) In metereology, precipitation is any product of the condensation of atmo-
 9815 spheric water vapor that falls under gravity.
 9816 Q: What causes precipitation to fall? A: gravity

9817 In both WikiQA and SQuAD, the original texts are Wikipedia articles, and the ques-
 9818 tions are generated by crowdworkers.

9819 **Methods**

9820 A baseline method is to search the text for sentences or short passages that overlap with
 9821 both the query and the candidate answer (Richardson et al., 2013). In example (17.15), this
 9822 baseline would select the correct answer, since *James* appears in a sentence that includes
 9823 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:

$$\mathbf{u} = [\overrightarrow{\mathbf{h}}_{M_q}^{(q)}; \overleftarrow{\mathbf{h}}_0^{(q)}]. \quad [17.25]$$

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,

$$\hat{c} = \operatorname{argmax}_c \mathbf{o} \cdot \mathbf{x}_c. \quad [17.29]$$

9824 This architecture can be trained end-to-end from a loss based on the log-likelihood of the
 9825 correct answer. A number of related architectures have been proposed (e.g., Hermann
 9826 et al., 2015; Kadlec et al., 2016; Dhingra et al., 2017; Cui et al., 2017), and these methods are
 9827 surveyed by Wang et al. (2017).

9828 Additional resources

9829 The field of information extraction is surveyed in course notes by Grishman (2012), and
 9830 more recently in a short survey paper (Grishman, 2015). Shen et al. (2015) survey the task
 9831 of entity linking, and Ji and Grishman (2011) survey work on knowledge base popula-
 9832 tion. This chapter’s discussion of non-propositional meaning was strongly influenced by
 9833 Morante and Sporleder (2012), who introduced a special issue of the journal *Computational*
 9834 *Linguistics* dedicated to recent work on modality and negation.

9835 Exercises

- 9836 1. Go to the Wikipedia page for your favorite movie. For each record in the info box
 9837 (e.g., *Screenplay by: Stanley Kubrick*), report whether there is a sentence in the ar-
 9838 ticle containing both the field and value (e.g., *The screenplay was written by Stanley*
 9839 *Kubrick*). If not, is there a sentence in the article containing just the value? (For
 9840 records with more than one value, just use the first value.)
- 9841 2. Building on your answer in the previous question, report the dependency path be-
 9842 tween the head words of the field and value for at least three records.
- 9843 3. Consider the following heuristic for entity linking:
 - 9844 • Among all entities that have the same type as the mention (e.g., LOC, PER),
 9845 choose the one whose name has the lowest edit distance from the mention.
 - 9846 • If more than one entity has the right type and the lowest edit distance from the
 9847 mention, choose the most popular one.
 - 9848 • If no candidate entity has the right type, choose NIL.

Now suppose you have the following feature function:

$$\mathbf{f}(y, \mathbf{x}) = [\operatorname{edit-dist}(\operatorname{name}(y), \mathbf{x}), \operatorname{same-type}(y, \mathbf{x}), \operatorname{popularity}(y), \delta(y = \text{NIL})]$$

9849 Design a set of ranking weights θ that match the heuristic. You may assume that
 9850 edit distance and popularity are always in the range [0, 100], and that the NIL entity
 9851 has values of zero for all features except δ ($y = \text{NIL}$).

9852 4. Now consider another heuristic:

- 9853 • Among all candidate entities that have edit distance zero from the mention,
 9854 and are the right type, choose the most popular one.
- 9855 • If no entity has edit distance zero from the mention, choose the one with the
 9856 right type that is most popular, regardless of edit distance.
- 9857 • If no entity has the right type, choose NIL.

9858 Using the same features and assumptions from the previous problem, prove that
 9859 there is no set of weights that could implement this heuristic. Then show that the
 9860 heuristic can be implemented by adding a single feature. Your new feature should
 9861 consider only the edit distance.

9862 5. Download the Reuters corpus in NLTK, and iterate over the tokens in the corpus:

```
9863 import nltk
9864 nltk.corpus.download('reuters')
9865 from nltk.corpus import reuters
9866 for word in reuters.words():
9867     #your code here
```

9868 a) Apply the pattern *_____*, such as *_____* to obtain candidates for the IS-A relation,
 9869 e.g. IS-A(ROMANIA, COUNTRY). What are three pairs that this method identi-
 9870 fies correctly? What are three different pairs that it gets wrong?

9871 b) Design a pattern for the PRESIDENT relation, e.g. PRESIDENT(PHILIPPINES, CORAZON AQUINO)
 9872 In this case, you may want to augment your pattern matcher with the ability
 9873 to match multiple token wildcards, perhaps using case information to detect
 9874 proper names. Again, list three correct

9875 c) Preprocess the Reuters data by running a named entity recognizer, replacing
 9876 tokens with named entity spans when applicable — e.g., your pattern can now
 9877 match on *the United States* if the NER system tags it. Apply your PRESIDENT
 9878 matcher to this preprocessed data. Does the accuracy improve? Compare 20
 9879 randomly-selected pairs from this pattern and the one you designed in the pre-
 9880 vious part.

9881 6. Using the same NLTK Reuters corpus, apply distant supervision to build a training
 9882 set for detecting the relation between nations and their capitals. Start with the fol-
 9883 lowing known relations: (JAPAN, TOKYO), (FRANCE, PARIS), (ITALY, ROME). How
 9884 many positive and negative examples are you able to extract?

- 9885 7. Represent the dependency path $\mathbf{x}^{(i)}$ as a sequence of words and dependency arcs
 9886 of length M_i , ignoring the endpoints of the path. In example 1 of Table 17.2, the
 9887 dependency path is,

$$\mathbf{x}^{(1)} = (\xleftarrow[\text{NSUBJ}]{} \text{traveled}, \xrightarrow[\text{OBL}]{}) \quad [17.30]$$

9888 If $x_m^{(i)}$ is a word, then let $\text{pos}(x_m^{(i)})$ be its part-of-speech, using the tagset defined in
 9889 chapter 8.

We can define the following kernel function over pairs of dependency paths (Bunescu and Mooney, 2005):

$$\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \begin{cases} 0, & M_i \neq M_j \\ \prod_{m=1}^{M_i} c(x_m^{(i)}, x_m^{(j)}), & M_i = M_j \end{cases}$$

$$c(x_m^{(i)}, x_m^{(j)}) = \begin{cases} 2, & x_m^{(i)} = x_m^{(j)} \\ 1, & x_m^{(i)} \neq x_m^{(j)} \text{ and } \text{pos}(x_m^{(i)}) = \text{pos}(x_m^{(j)}) \\ 0, & \text{otherwise.} \end{cases}$$

9890 Using this kernel function, compute the kernel similarities of example 1 from Ta-
 9891 ble 17.2 with the other five examples.

8. Continuing from the previous problem, suppose that the instances have the following labels:

$$y_2 = 1, y_3 = -1, y_4 = -1, y_5 = 1, y_6 = 1 \quad [17.31]$$

9892 Equation 17.13 defines a kernel-based classification in terms of parameters α and
 9893 b . Using the above labels for y_2, \dots, y_6 , identify the values of α and b under which
 9894 $\hat{y}_1 = 1$. Remember the constraint that $\alpha_i \geq 0$ for all i .

- 9895 9. Consider the neural QA system described in § 17.5.2, but restrict the set of candidate
 9896 answers to words in the passage, and set each candidate answer embedding \mathbf{x} equal
 9897 to the vector $\mathbf{h}_m^{(p)}$, representing token m in the passage, so that $\hat{m} = \text{argmax}_m \mathbf{o} \cdot \mathbf{h}_m^{(p)}$.
 9898 Suppose the system selects answer \hat{m} , but the correct answer is m^* . Consider the
 9899 gradient of the margin loss with respect to the attention:

9900 a) Prove that $\frac{\partial \ell}{\partial \alpha_{\hat{m}}} \geq \frac{\partial \ell}{\partial \alpha_{m^*}}$.

9901 b) Assuming that $\|\mathbf{h}_{\hat{m}}\| = \|\mathbf{h}_{m^*}\|$, prove that $\frac{\partial \ell}{\partial \alpha_{\hat{m}}} \geq 0$ and $\frac{\partial \ell}{\partial \alpha_{m^*}} \leq 0$. Explain in
 9902 words what this means about how the attention is expected to change after a
 9903 gradient-based update.

9904

Chapter 18

9905

Machine translation

9906 Machine translation (MT) is one of the “holy grail” problems in artificial intelligence,
9907 with the potential to transform society by facilitating communication between people
9908 anywhere in the world. As a result, MT has received significant attention and funding
9909 since the early 1950s. However, it has proved remarkably challenging, and while there
9910 has been substantial progress towards usable MT systems — especially for high-resource
9911 language pairs like English-French — we are still far from translation systems that match
9912 the nuance and depth of human translations.

9913

18.1 Machine translation as a task

9914 Machine translation can be formulated as an optimization problem:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w}^{(t)}}{\operatorname{argmax}} \Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}), \quad [18.1]$$

9915 where $\mathbf{w}^{(s)}$ is a sentence in a **source** language, $\mathbf{w}^{(t)}$ is a sentence in the **target language**,
9916 and Ψ is a scoring function. As usual, this formalism requires two components: a decod-
9917 ing algorithm for computing $\hat{\mathbf{w}}^{(t)}$, and a learning algorithm for estimating the parameters
9918 of the scoring function Ψ .

9919 Decoding is difficult for machine translation because of the huge space of possible
9920 translations. We have faced large label spaces before: for example, in sequence labeling,
9921 the set of possible label sequences is exponential in the length of the input. In these cases,
9922 it was possible to search the space quickly by introducing locality assumptions: for ex-
9923 ample, that each tag depends only on its predecessor, or that each production depends
9924 only on its parent. In machine translation, no such locality assumptions seem possible:
9925 human translators reword, reorder, and rearrange words; they replace single words with
9926 multi-word phrases, and vice versa. This flexibility means that in even relatively simple

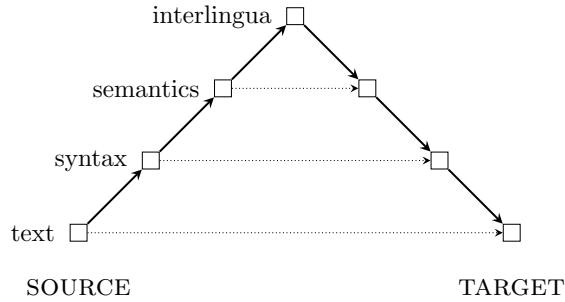


Figure 18.1: The Vauquois Pyramid

9927 translation models, decoding is NP-hard (Knight, 1999). Approaches for dealing with this
 9928 complexity are described in § 18.4.

Estimating translation models is difficult as well. Labeled translation data usually comes in the form parallel sentences, e.g.,

$$\begin{aligned} \mathbf{w}^{(s)} &= A \text{ Vinay le gusta las manzanas.} \\ \mathbf{w}^{(t)} &= \text{Vinay likes apples.} \end{aligned}$$

9929 A useful feature function would note the translation pairs (*gusta*, *likes*), (*manzanas*, *apples*),
 9930 and even (*Vinay*, *Vinay*). But this word-to-word **alignment** is not given in the data. One
 9931 solution is to treat this alignment as a **latent variable**; this is the approach taken by clas-
 9932 sical **statistical machine translation** (SMT) systems, described in § 18.2. Another solution
 9933 is to model the relationship between $\mathbf{w}^{(t)}$ and $\mathbf{w}^{(s)}$ through a more complex and expres-
 9934 sive function; this is the approach taken by **neural machine translation** (NMT) systems,
 9935 described in § 18.3.

9936 The **Vauquois Pyramid** is a theory of how translation should be done. At the lowest
 9937 level, the translation system operates on individual words, but the horizontal distance
 9938 at this level is large, because languages express ideas differently. If we can move up the
 9939 triangle to syntactic structure, the distance for translation is reduced; we then need only
 9940 produce target-language text from the syntactic representation, which can be as simple
 9941 as reading off a tree. Further up the triangle lies semantics; translating between semantic
 9942 representations should be easier still, but mapping between semantics and surface text is a
 9943 difficult, unsolved problem. At the top of the triangle is **interlingua**, a semantic represen-
 9944 tation that is so generic that it is identical across all human languages. Philosophers de-
 9945 bate whether such a thing as interlingua is really possible (e.g., Derrida, 1985). While the
 9946 first-order logic representations discussed in chapter 12 might be thought to be language
 9947 independent, they are built on an inventory of predicates that are suspiciously similar to
 9948 English words (Nirenburg and Wilks, 2001). Nonetheless, the idea of linking translation

| | 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*.

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 reference translation is $w^{(t)} = Vinay likes Python$. However, the **gloss**, or word-for-word translation $w^{(t)} = To Vinay it like Python$ is also 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.

The BLEU score is then based on the average, $\exp \frac{1}{N} \sum_{n=1}^N \log p_n$. Two modifications of Equation 18.2 are necessary: (1) to avoid computing $\log 0$, all precisions are smoothed to ensure that they are positive; (2) each n -gram in the reference can be used at most once, so that *to to to to* does not achieve $p_1 = 1$ against the reference *to be or not to be*. Furthermore, precision-based metrics are biased in favor of short translations, which

| | Translation | p_1 | p_2 | p_3 | p_4 | BP | BLEU |
|------------------|---|---------------|---------------|---------------|---------------|-----|------|
| <i>Reference</i> | <i>Vinay likes programming in Python</i> | | | | | | |
| <i>Sys1</i> | <i>To Vinay it like to program Python</i> | $\frac{2}{7}$ | 0 | 0 | 0 | 1 | .21 |
| <i>Sys2</i> | <i>Vinay likes Python</i> | $\frac{3}{3}$ | $\frac{1}{2}$ | 0 | 0 | .51 | .33 |
| <i>Sys3</i> | <i>Vinay likes programming in his pajamas</i> | $\frac{4}{6}$ | $\frac{3}{5}$ | $\frac{2}{4}$ | $\frac{1}{3}$ | 1 | .76 |

Figure 18.2: A reference translation and three system outputs. For each output, p_n indicates the precision at each n -gram, and BP indicates the brevity penalty.

can achieve high scores by minimizing the denominator in [18.2]. To avoid this issue, a **brevity penalty** is applied to translations that are shorter than the reference. This penalty is indicated as “BP” in Figure 18.2.

Automated metrics like BLEU have been validated by correlation with human judgments of translation quality. Nonetheless, it is not difficult to construct examples in which the BLEU score is high, yet the translation is disfluent or carries a completely different meaning from the original. To give just one example, consider the problem of translating pronouns. Because pronouns refer to specific entities, a single incorrect pronoun can obliterate the semantics of the original sentence. Existing state-of-the-art systems generally do not attempt the reasoning necessary to correctly resolve pronominal anaphora (Hardmeier, 2012). Despite the importance of pronouns for semantics, they have a marginal impact on BLEU, which may help to explain why existing systems do not make a greater effort to translate them correctly.

Fairness and bias The problem of pronoun translation intersects with issues of fairness and bias. In many languages, such as Turkish, the third person singular pronoun is gender neutral. Today’s state-of-the-art systems produce the following Turkish-English translations (Caliskan et al., 2017):

- 9990 (18.1) *O bir doktor.*
He is a doctor.
- 9991 (18.2) *O bir hemşire.*
She is a nurse.

The same problem arises for other professions that have stereotypical genders, such as engineers, soldiers, and teachers, and for other languages that have gender-neutral pronouns. This bias was not directly programmed into the translation model; it arises from statistical tendencies in existing datasets. This highlights a general problem with data-driven approaches, which can perpetuate biases that negatively impact disadvantaged

9997 groups. Worse, machine learning can *amplify* biases in data (Bolukbasi et al., 2016): if a
9998 dataset has even a slight tendency towards men as doctors, the resulting translation model
9999 may produce translations in which doctors are always *he*, and nurses are always *she*.

10000 **Other metrics** A range of other automated metrics have been proposed for machine
10001 translation. One potential weakness of BLEU is that it only measures precision; METEOR
10002 is a weighted *F*-MEASURE, which is a combination of recall and precision (see § 4.4.1).
10003 **Translation Error Rate (TER)** computes the string **edit distance** (see § 9.1.4) between the
10004 reference and the hypothesis (Snover et al., 2006). For language pairs like English and
10005 Japanese, there are substantial differences in word order, and word order errors are not
10006 sufficiently captured by *n*-gram based metrics. The **RIBES** metric applies rank correla-
10007 tion to measure the similarity in word order between the system and reference transla-
10008 tions (Isozaki et al., 2010).

10009 18.1.2 Data

10010 Data-driven approaches to machine translation rely primarily on **parallel corpora**, which
10011 are translations at the sentence level. Early work focused on government records, in which
10012 fine-grained official translations are often required. For example, the IBM translation sys-
10013 tems were based on the proceedings of the Canadian Parliament, called **Hansards**, which
10014 are recorded in English and French (Brown et al., 1990). The growth of the European
10015 Union led to the development of the **EuroParl corpus**, which spans 21 European lan-
10016 guages (Koehn, 2005). While these datasets helped to launch the field of machine transla-
10017 tion, they are restricted to narrow domains and a formal speaking style, limiting their ap-
10018 plicability to other types of text. As more resources are committed to machine translation,
10019 new translation datasets have been commissioned. This has broadened the scope of avail-
10020 able data to news,¹ movie subtitles,² social media (Ling et al., 2013), dialogues (Fordyce,
10021 2007), TED talks (Paul et al., 2010), and scientific research articles (Nakazawa et al., 2016).

10022 Despite this growing set of resources, the main bottleneck in machine translation data
10023 is the need for parallel corpora that are aligned at the sentence level. Many languages have
10024 sizable parallel corpora with some high-resource language, but not with each other. The
10025 high-resource language can then be used as a “pivot” or “bridge” (Boitet, 1988; Utiyama
10026 and Isahara, 2007): for example, De Gispert and Marino (2006) use Spanish as a bridge for
10027 translation between Catalan and English. For most of the 6000 languages spoken today,
10028 the only source of translation data remains the Judeo-Christian Bible (Resnik et al., 1999).
10029 While relatively small, at less than a million tokens, the Bible has been translated into
10030 more than 2000 languages, far outpacing any other corpus. Some research has explored

¹https://catalog.ldc.upenn.edu/LDC2010T10_translation-task.html ²<http://opus.nlpl.eu/> <http://www.statmt.org/wmt15/>

10031 the possibility of automatically identifying parallel sentence pairs from unaligned parallel
 10032 texts, such as web pages and Wikipedia articles (Kilgarriff and Grefenstette, 2003; Resnik
 10033 and Smith, 2003; Adafre and De Rijke, 2006). Another approach is to create large parallel
 10034 corpora through crowdsourcing (Zaidan and Callison-Burch, 2011).

10035 18.2 Statistical machine translation

10036 The previous section introduced adequacy and fluency as the two main criteria for ma-
 10037 chine translation. A natural modeling approach is to represent them with separate scores,

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) + \Psi_F(\mathbf{w}^{(t)}). \quad [18.3]$$

10038 The fluency score Ψ_F need not even consider the source sentence; it only judges $\mathbf{w}^{(t)}$ on
 10039 whether it is fluent in the target language. This decomposition is advantageous because
 10040 it makes it possible to estimate the two scoring functions on separate data. While the
 10041 adequacy model must be estimated from aligned sentences — which are relatively expen-
 10042 sive and rare — the fluency model can be estimated from monolingual text in the target
 10043 language. Large monolingual corpora are now available in many languages, thanks to
 10044 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:

$$\Psi_A(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \triangleq \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) \quad [18.4]$$

$$\Psi_F(\mathbf{w}^{(t)}) \triangleq \log p_T(\mathbf{w}^{(t)}) \quad [18.5]$$

$$\Psi(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \log p_{S|T}(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}) + \log p_T(\mathbf{w}^{(t)}) = \log p_{S,T}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}). \quad [18.6]$$

10045 By setting the scoring functions equal to the logarithms of the prior and likelihood, their
 10046 sum is equal to $\log p_{S,T}$, which is the logarithm of the joint probability of the source and
 10047 target. The sentence $\hat{\mathbf{w}}^{(t)}$ that maximizes this joint probability is also the maximizer of the
 10048 conditional probability $p_{T|S}$, making it the most likely target language sentence, condi-
 10049 tioned on the source.

10050 The noisy channel model can be justified by a generative story. The target text is orig-
 10051 inally generated from a probability model p_T . It is then encoded in a “noisy channel”
 10052 $p_{S|T}$, which converts it to a string in the source language. In decoding, we apply Bayes’
 10053 rule to recover the string $\mathbf{w}^{(t)}$ that is maximally likely under the conditional probability
 10054 $p_{T|S}$. Under this interpretation, the target probability p_T is just a language model, and
 10055 can be estimated using any of the techniques from chapter 6. The only remaining learning
 10056 problem is to estimate the translation model $p_{S|T}$.

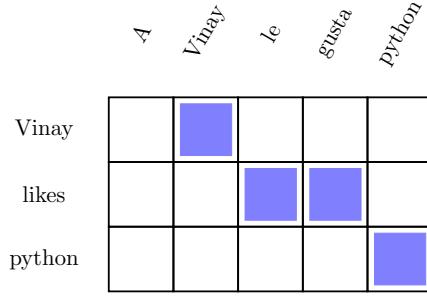


Figure 18.3: An example word-to-word alignment

18.2.1 Statistical translation modeling

The simplest decomposition of the translation model is word-to-word: each word in the source should be aligned to a word in the translation. This approach presupposes an **alignment** $\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)})$, which contains a list of pairs of source and target tokens. For example, given $\mathbf{w}^{(s)} = A\ Vinay\ le\ gusta\ Python$ and $\mathbf{w}^{(t)} = Vinay\ likes\ Python$, one possible word-to-word alignment is,

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, \emptyset), (Vinay, Vinay), (le, likes), (gusta, likes), (Python, Python)\}. \quad [18.7]$$

This alignment is shown in Figure 18.3. Another, less promising, alignment is:

$$\mathcal{A}(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \{(A, Vinay), (Vinay, likes), (le, Python), (gusta, \emptyset), (Python, \emptyset)\}. \quad [18.8]$$

Each alignment contains exactly one tuple for each word in the *source*, which serves to explain how the source word could be translated from the target, as required by the translation probability $p_{S|T}$. If no appropriate word in the target can be identified for a source word, it is aligned to \emptyset — as is the case for the Spanish function word *a* in the example, which glosses to the English word *to*. Words in the target can align with multiple words in the source, so that the target word *likes* can align to both *le* and *gusta* in the source.

The joint probability of the alignment and the translation can be defined conveniently as,

$$p(\mathbf{w}^{(s)}, \mathcal{A} | \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)}, a_m | w_{a_m}^{(t)}, m, M^{(s)}, M^{(t)}) \quad [18.9]$$

$$= \prod_{m=1}^{M^{(s)}} p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)}). \quad [18.10]$$

This probability model makes two key assumptions:

- 10071 • The alignment probability factors across tokens,

$$p(\mathcal{A} \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m \mid m, M^{(s)}, M^{(t)}). \quad [18.11]$$

10072 This means that each alignment decision is independent of the others, and depends
 10073 only on the index m , and the sentence lengths $M^{(s)}$ and $M^{(t)}$.

- 10074 • The translation probability also factors across tokens,

$$p(\mathbf{w}^{(s)} \mid \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{m=1}^{M^{(s)}} p(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.12]$$

10075 so that each word in $\mathbf{w}^{(s)}$ depends only on its aligned word in $\mathbf{w}^{(t)}$. This means that
 10076 translation is word-to-word, ignoring context. The hope is that the target language
 10077 model $p(\mathbf{w}^{(t)})$ will correct any disfluencies that arise from word-to-word translation.

To translate with such a model, we could sum or max over all possible alignments,

$$p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \sum_{\mathcal{A}} p(\mathbf{w}^{(s)}, \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.13]$$

$$= p(\mathbf{w}^{(t)}) \sum_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} \mid \mathbf{w}^{(t)}, \mathcal{A}) \quad [18.14]$$

$$\geq p(\mathbf{w}^{(t)}) \max_{\mathcal{A}} p(\mathcal{A}) \times p(\mathbf{w}^{(s)} \mid \mathbf{w}^{(t)}, \mathcal{A}). \quad [18.15]$$

The term $p(\mathcal{A})$ defines the prior probability over alignments. A series of alignment models with increasingly relaxed independence assumptions was developed by researchers at IBM in the 1980s and 1990s, known as IBM Models 1-6 (Och and Ney, 2003). IBM Model 1 makes the strongest independence assumption:

$$p(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}. \quad [18.16]$$

10078 In this model, every alignment is equally likely. This is almost surely wrong, but it re-
 10079 sults in a convex learning objective, yielding a good initialization for the more complex
 10080 alignment models (Brown et al., 1993; Koehn, 2009).

10081 18.2.2 Estimation

10082 Let us define the parameter $\theta_{u \rightarrow v}$ as the probability of translating target word u to source
 10083 word v . If word-to-word alignments were annotated, these probabilities could be com-
 10084 puted from relative frequencies,

$$\hat{\theta}_{u \rightarrow v} = \frac{\text{count}(u, v)}{\text{count}(u)}, \quad [18.17]$$

10085 where $\text{count}(u, v)$ is the count of instances in which word v was aligned to word u in
 10086 the training set, and $\text{count}(u)$ is the total count of the target word u . The smoothing
 10087 techniques mentioned in chapter 6 can help to reduce the variance of these probability
 10088 estimates.

10089 Conversely, if we had an accurate translation model, we could estimate the likelihood
 10090 of each alignment decision,

$$q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \propto p(a_m \mid m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} \mid w_{a_m}^{(t)}), \quad [18.18]$$

where $q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)})$ is a measure of our confidence in aligning source word $w_m^{(s)}$
 to target word $w_{a_m}^{(t)}$. The relative frequencies could then be computed from the *expected
 counts*,

$$\hat{\theta}_{u \rightarrow v} = \frac{E_q [\text{count}(u, v)]}{\text{count}(u)} \quad [18.19]$$

$$E_q [\text{count}(u, v)] = \sum_m q_m(a_m \mid \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) \times \delta(w_m^{(s)} = v) \times \delta(w_{a_m}^{(t)} = u). \quad [18.20]$$

10091 The **expectation-maximization** (EM) algorithm proceeds by iteratively updating q_m
 10092 and $\hat{\Theta}$. The algorithm is described in general form in chapter 5. For statistical machine
 10093 translation, the steps of the algorithm are:

- 10094 1. **E-step:** Update beliefs about word alignment using Equation 18.18.
 10095 2. **M-step:** Update the translation model using Equations 18.19 and 18.20.

10096 As discussed in chapter 5, the expectation maximization algorithm is guaranteed to con-
 10097 verge, but not to a global optimum. However, for IBM Model 1, it can be shown that EM
 10098 optimizes a convex objective, and global optimality is guaranteed. For this reason, IBM
 10099 Model 1 is often used as an initialization for more complex alignment models. For more
 10100 detail, see Koehn (2009).

10101 18.2.3 Phrase-based translation

10102 Real translations are not word-to-word substitutions. One reason is that many multiword
 10103 expressions are not translated literally, as shown in this example from French:

- 10104 (18.3) *Nous allons prendre un verre*
 We will take a glass
 10105 We'll have a drink

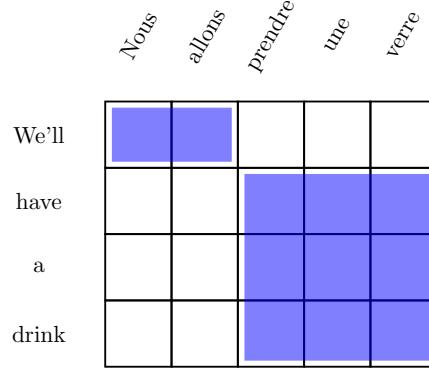


Figure 18.4: A phrase-based alignment between French and English, corresponding to example (18.3)

10106 The line *we will take a glass* is the word-for-word gloss of the French sentence; the transla-
 10107 tion *we'll have a drink* is shown on the third line. Such examples are difficult for word-to-
 10108 word translation models, since they require translating *prendre* to *have* and *verre* to *drink*.
 10109 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

$$p(\mathbf{w}^{(s)} | \mathbf{w}^{(t)}, \mathcal{A}) = \prod_{((i,j),(k,\ell)) \in \mathcal{A}} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \dots, w_j^{(s)}\} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \dots, w_\ell^{(t)}\}). \quad [18.21]$$

10110 The phrase alignment $((i, j), (k, \ell))$ indicates that the span $\mathbf{w}_{i+1:j}^{(s)}$ is the translation of the
 10111 span $\mathbf{w}_{k+1:\ell}^{(t)}$. An example phrasal alignment is shown in Figure 18.4. Note that the align-
 10112 ment set \mathcal{A} is required to cover all of the tokens in the source, just as in word-based trans-
 10113 lation. The probability model $p_{w^{(s)}|w^{(t)}}$ must now include translations for all phrase pairs,
 10114 which can be learned from expectation-maximization just as in word-based statistical ma-
 10115 chine translation.

10116 **18.2.4 *Syntax-based translation**

10117 The Vauquois Pyramid (Figure 18.1) suggests that translation might be easier if we take a
 10118 higher-level view. One possibility is to incorporate the syntactic structure of the source,
 10119 the target, or both. This is particularly promising for language pairs that consistent syn-
 10120 tactic differences. For example, English adjectives almost always precede the nouns that
 10121 they modify, while in Romance languages such as French and Spanish, the adjective often
 10122 follows the noun: thus, *angry fish* would translate to *pez (fish) enojado (angry)* in Spanish.
 10123 In word-to-word translation, these reorderings cause the alignment model to be overly
 10124 permissive. It is not that the order of *any* pair of English words can be reversed when
 10125 translating into Spanish, but only adjectives and nouns within a noun phrase. Similar
 10126 issues arise when translating between verb-final languages such as Japanese (in which
 10127 verbs usually follow the subject and object), verb-initial languages like Tagalog and clas-
 10128 sical Arabic, and verb-medial languages such as English.

10129 An elegant solution is to link parsing and translation in a **synchronous context-free**
 10130 **grammar** (SCFG; Chiang, 2007).³ An SCFG is a set of productions of the form $X \rightarrow (\alpha, \beta, \sim)$,
 10131 where X is a non-terminal, α and β are sequences of terminals or non-terminals, and \sim
 10132 is a one-to-one alignment of items in α with items in β . To handle the English-Spanish
 10133 adjective-noun ordering, an SCFG would include productions such as,

$$\text{NP} \rightarrow (\text{DET}_1 \text{NN}_2 \text{JJ}_3, \quad \text{DET}_1 \text{JJ}_3 \text{NN}_2), \quad [18.22]$$

10134 with subscripts indicating the alignment between the Spanish (left) and English (right)
 10135 parts of the right-hand side. Terminal productions yield translation pairs,

$$\text{JJ} \rightarrow (\text{enojado}_1, \text{angry}_1). \quad [18.23]$$

10136 A synchronous derivation begins with the start symbol S , and derives a pair of sequences
 10137 of terminal symbols.

10138 Given an SCFG in which each production yields at most two symbols in each lan-
 10139 guage (Chomsky Normal Form; see § 9.2.1), a sentence can be parsed using only the CKY
 10140 algorithm (chapter 10). The resulting derivation also includes productions in the other
 10141 language, all the way down to the surface form. Therefore, SCFGs make translation very
 10142 similar to parsing. In a weighted SCFG, the log probability $\log p_{S|T}$ can be computed from
 10143 the sum of the log-probabilities of the productions. However, combining SCFGs with a
 10144 target language model is computationally expensive, necessitating approximate search
 10145 algorithms (Huang and Chiang, 2007).

10146 Synchronous context-free grammars are an example of **tree-to-tree translation**, be-
 10147 cause they model the syntactic structure of both the target and source language. In **string-**
 10148 **to-tree translation**, string elements are translated into constituent tree fragments, which

³Earlier approaches to syntactic machine translation includes syntax-driven transduction (Lewis II and Stearns, 1968) and stochastic inversion grammars (Wu, 1997).

10149 are then assembled into a translation (Yamada and Knight, 2001; Galley et al., 2004); in
 10150 **tree-to-string translation**, the source side is parsed, and then transformed into a string on
 10151 the target side (Liu et al., 2006). A key question for syntax-based translation is the extent
 10152 to which we phrasal constituents align across translations (Fox, 2002), because this gov-
 10153 erns the extent to which we can rely on monolingual parsers and treebanks. For more on
 10154 syntax-based machine translation, see the monograph by Williams et al. (2016).

10155 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.

$$\mathbf{z} = \text{ENCODE}(\mathbf{w}^{(s)}) \quad [18.24]$$

$$\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)} \sim \text{DECODE}(\mathbf{z}), \quad [18.25]$$

10156 where the second line means that the function $\text{DECODE}(\mathbf{z})$ defines the conditional proba-
 10157 bility $p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)})$.

The decoder is typically a recurrent neural network, which generates the target language sentence one word at a time, while recurrently updating a hidden state. The encoder and decoder networks are trained end-to-end from parallel sentences. If the output layer of the decoder is a logistic function, then the entire architecture can be trained to maximize the conditional log-likelihood,

$$\log p(\mathbf{w}^{(t)} \mid \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.26]$$

$$p(w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)}) \propto \exp\left(\boldsymbol{\beta}_{w_m^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}\right) \quad [18.27]$$

where the hidden state $\mathbf{h}_{m-1}^{(t)}$ is a recurrent function of the previously generated text $\mathbf{w}_{1:m-1}^{(t)}$ and the encoding \mathbf{z} . The second line is equivalent to writing,

$$w_m^{(t)} \mid \mathbf{w}_{1:m-1}^{(t)}, \mathbf{w}^{(s)} \sim \text{SoftMax}\left(\boldsymbol{\beta} \cdot \mathbf{h}_{m-1}^{(t)}\right), \quad [18.28]$$

10158 where $\boldsymbol{\beta} \in \mathbb{R}^{(V^{(t)} \times K)}$ is the matrix of output word vectors for the $V^{(t)}$ words in the target
 10159 language vocabulary.

The simplest encoder-decoder architecture is the **sequence-to-sequence model** (Sutskever et al., 2014). In this model, the encoder is set to the final hidden state of a **long short-term**

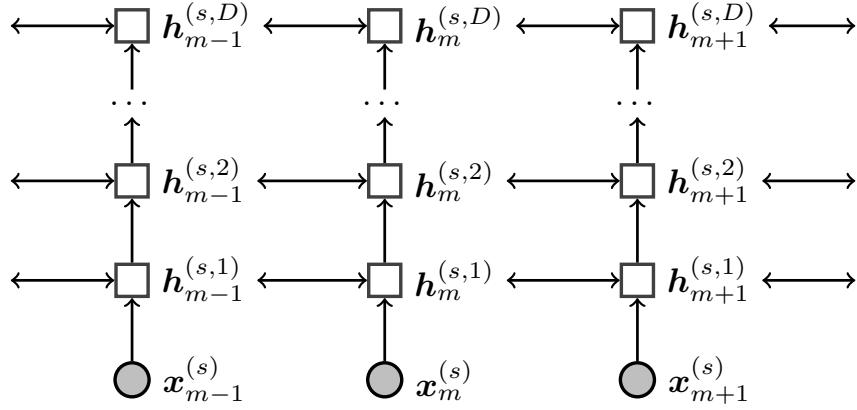


Figure 18.5: A deep bidirectional LSTM encoder

memory (LSTM) (see § 6.3.3) on the source sentence:

$$\mathbf{h}_m^{(s)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.29]$$

$$\mathbf{z} \triangleq \mathbf{h}_{M^{(s)}}^{(s)}, \quad [18.30]$$

where $\mathbf{x}_m^{(s)}$ is the embedding of source language word $w_m^{(s)}$. The encoding then provides the initial hidden state for the decoder LSTM:

$$\mathbf{h}_0^{(t)} = \mathbf{z} \quad [18.31]$$

$$\mathbf{h}_m^{(t)} = \text{LSTM}(\mathbf{x}_m^{(t)}, \mathbf{h}_{m-1}^{(t)}), \quad [18.32]$$

10160 where $\mathbf{x}_m^{(t)}$ is the embedding of the target language word $w_m^{(t)}$.

10161 Sequence-to-sequence translation is nothing more than wiring together two LSTMs:
 10162 one to read the source, and another to generate the target. To make the model work well,
 10163 some additional tweaks are needed:

- 10164 • Most notably, the model works much better if the source sentence is reversed, reading
 10165 from the end of the sentence back to the beginning. In this way, the words at the
 10166 beginning of the source have the greatest impact on the encoding \mathbf{z} , and therefore
 10167 impact the words at the beginning of the target sentence. Later work on more advanced
 10168 encoding models, such as **neural attention** (see § 18.3.1), has eliminated the
 10169 need for reversing the source sentence.
- The encoder and decoder can be implemented as **deep LSTMs**, with multiple layers of hidden states. As shown in Figure 18.5, each hidden state $\mathbf{h}_m^{(s,i)}$ at layer i is treated

as the input to an LSTM at layer $i + 1$:

$$\mathbf{h}_m^{(s,1)} = \text{LSTM}(\mathbf{x}_m^{(s)}, \mathbf{h}_{m-1}^{(s)}) \quad [18.33]$$

$$\mathbf{h}_m^{(s,i+1)} = \text{LSTM}(\mathbf{h}_m^{(s,i)}, \mathbf{h}_{m-1}^{(s,i+1)}), \quad \forall i \geq 1. \quad [18.34]$$

10170 The original work on sequence-to-sequence translation used four layers; in 2016,
 10171 Google’s commercial machine translation system used eight layers (Wu et al., 2016).⁴

- 10172 • Significant improvements can be obtained by creating an **ensemble** of translation
 10173 models, each trained from a different random initialization. For an ensemble of size
 10174 N , the per-token decoding probability is set equal to,

$$p(w^{(t)} | z, \mathbf{w}_{1:m-1}^{(t)}) = \frac{1}{N} \sum_{i=1}^N p_i(w^{(t)} | z, \mathbf{w}_{1:m-1}^{(t)}), \quad [18.35]$$

10175 where p_i is the decoding probability for model i . Each translation model in the
 10176 ensemble includes its own encoder and decoder networks.

- 10177 • The original sequence-to-sequence model used a fairly standard training setup: stochastic
 10178 gradient descent with an exponentially decreasing learning rate after the first five
 10179 epochs; mini-batches of 128 sentences, chosen to have similar length so that each
 10180 sentence on the batch will take roughly the same amount of time to process; gradient
 10181 clipping (see § 3.3.4) to ensure that the norm of the gradient never exceeds some
 10182 predefined value.

10183 18.3.1 Neural attention

10184 The sequence-to-sequence model discussed in the previous section was a radical departure
 10185 from statistical machine translation, in which each word or phrase in the target lan-
 10186 guage is conditioned on a single word or phrase in the source language. Both approaches
 10187 have advantages. Statistical translation leverages the idea of compositionality — transla-
 10188 tions of large units should be based on the translations of their component parts — and
 10189 this seems crucial if we are to scale translation to longer units of text. But the translation
 10190 of each word or phrase often depends on the larger context, and encoder-decoder models
 10191 capture this context at the sentence level.

10192 Is it possible for translation to be both contextualized and compositional? One ap-
 10193 proach is to augment neural translation with an **attention mechanism**. The idea of neural
 10194 attention was described in § 17.5, but its application to translation bears further discus-
 10195 sion. In general, attention can be thought of as using a query to select from a memory
 10196 of key-value pairs. However, the query, keys, and values are all vectors, and the entire

⁴Google reports that this system took six days to train for English-French translation, using 96 NVIDIA K80 GPUs, which would have cost roughly half a million dollars at the time.

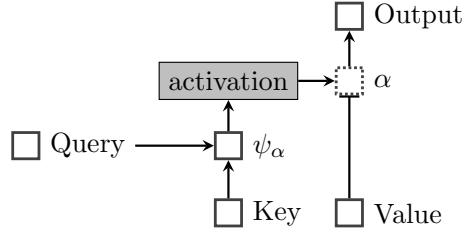


Figure 18.6: A general view of neural attention. The dotted box indicates that each $\alpha_{m \rightarrow n}$ can be viewed as a **gate** on value n .

operation is differentiable. For each key n in the memory, we compute a score $\psi_\alpha(m, n)$ with respect to the query m . That score is a function of the compatibility of the key and the query, and can be computed using a small feedforward neural network. The vector of scores is passed through an activation function, such as softmax. The output of this activation function is a vector of non-negative numbers $[\alpha_{m \rightarrow 1}, \alpha_{m \rightarrow 2}, \dots, \alpha_{m \rightarrow N}]^\top$, with length N equal to the size of the memory. Each value in the memory v_n is multiplied by the attention $\alpha_{m \rightarrow n}$; the sum of these scaled values is the output. This process is shown in Figure 18.6. In the extreme case that $\alpha_{m \rightarrow n} = 1$ and $\alpha_{m \rightarrow n'} = 0$ for all other n' , then the attention mechanism simply selects the value v_n from the memory.

Neural attention makes it possible to integrate alignment into the encoder-decoder architecture. Rather than encoding the entire source sentence into a fixed length vector z , it can be encoded into a matrix $Z \in \mathbb{R}^{K \times M^{(S)}}$, where K is the dimension of the hidden state, and $M^{(S)}$ is the number of tokens in the source input. Each column of Z represents the state of a recurrent neural network over the source sentence. These vectors are constructed from a **bidirectional LSTM** (see § 7.6), which can be a deep network as shown in Figure 18.5. These columns are both the keys and the values in the attention mechanism.

At each step m in decoding, the attentional state is computed by executing a query, which is equal to the state of the decoder, $h_m^{(t)}$. The resulting compatibility scores are,

$$\psi_\alpha(m, n) = v_\alpha \cdot \tanh(\Theta_\alpha[h_m^{(t)}; h_n^{(s)}]). \quad [18.36]$$

The function ψ is thus a two layer feedforward neural network, with weights v_α on the output layer, and weights Θ_α on the input layer. To convert these scores into attention weights, we apply an activation function, which can be vector-wise softmax or an element-wise sigmoid:

Softmax attention

$$\alpha_{m \rightarrow n} = \frac{\exp \psi_\alpha(m, n)}{\sum_{n'=1}^{M^{(s)}} \exp \psi_\alpha(m, n')} \quad [18.37]$$

Sigmoid attention

$$\alpha_{m \rightarrow n} = \sigma(\psi_\alpha(m, n)) \quad [18.38]$$

The attention α is then used to compute a 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):

$$\tilde{h}_m^{(t)} = \tanh(\Theta_c[h_m^{(t)}; c_m]) \quad [18.40]$$

$$p(w_{m+1}^{(t)} | w_{1:m}^{(t)}, w^{(s)}) \propto \exp\left(\beta_{w_{m+1}^{(t)}} \cdot \tilde{h}_m^{(t)}\right). \quad [18.41]$$

10217 Here the decoder state $h_m^{(t)}$ is concatenated with the context vector, forming the input
 10218 to compute a final output vector $\tilde{h}_m^{(t)}$. The context vector can be incorporated into the
 10219 decoder recurrence in a similar manner (Bahdanau et al., 2014).

10220 **18.3.2 *Neural machine translation without recurrence**

In the encoder-decoder model, attention’s “keys and values” are the hidden state representations in the encoder network, z , and the “queries” are state representations in the decoder network $h^{(t)}$. It is also possible to completely eliminate recurrence from neural translation, by applying **self-attention** (Lin et al., 2017; Kim et al., 2017) within the encoder and decoder, as in the **transformer architecture** (Vaswani et al., 2017). For level i , the basic equations of the encoder side of the transformer are:

$$z_m^{(i)} = \sum_{n=1}^{M^{(s)}} \alpha_{m \rightarrow n}^{(i)} (\Theta_v h_n^{(i-1)}) \quad [18.42]$$

$$h_m^{(i)} = \Theta_2 \text{ReLU}\left(\Theta_1 z_m^{(i)} + b_1\right) + b_2. \quad [18.43]$$

10221 For each token m at level i , we compute self-attention over the entire source sentence:
 10222 the keys, values, and queries are all projections of the vector $h^{(i-1)}$. The attention scores
 10223 $\alpha_{m \rightarrow n}^{(i)}$ are computed using a scaled form of softmax attention,

$$\alpha_{m \rightarrow n} \propto \exp(\psi_\alpha(m, n)/M), \quad [18.44]$$

10224 where M is the length of the input. This encourages the attention to be more evenly
 10225 dispersed across the input. Self-attention is applied across multiple “heads”, each using
 10226 different projections of $\mathbf{h}^{(i-1)}$ to form the keys, values, and queries.

The output of the self-attentional layer is the representation $\mathbf{z}_m^{(i)}$, which is then passed through a two-layer feed-forward network, yielding the input to the next layer, $\mathbf{h}^{(i)}$. To ensure that information about word order in the source is integrated into the model, the encoder includes **positional encodings** of the index of each word in the source. These encodings are vectors for each position $m \in \{1, 2, \dots, M\}$. The transformer sets these encodings equal to a set of sinusoidal functions of m ,

$$e_{2i-1}(m) = \sin(m/(10000^{\frac{2i}{K_e}})) \quad [18.45]$$

$$e_{2i}(m) = \cos(m/(10000^{\frac{2i}{K_e}})), \quad \forall i \in \{1, 2, \dots, K_e/2\} \quad [18.46]$$

10227 where $e_{2i}(m)$ is the value at position $2i$ of the encoding for position m . As we progress
 10228 through the dimensions of the encoding, we encounter sinusoidal functions of progres-
 10229 sively wider bandwidth. This enables the model to learn to attend by relative positions of
 10230 words. The positional encodings are concatenated with the word embeddings \mathbf{x}_m at the
 10231 base layer of the model.⁵

10232 **Convolutional neural networks** (see § 3.4) have also been applied as encoders in neu-
 10233 ral machine translation. For each word $w_m^{(s)}$, a convolutional network computes a rep-
 10234 resentation $\mathbf{h}_m^{(s)}$ from the embeddings of the word and its neighbors. This procedure is
 10235 applied several times, creating a deep convolutional network. The recurrent decoder then
 10236 computes a set of attention weights over these convolutional representations, using the
 10237 decoder’s hidden state $\mathbf{h}^{(t)}$ as the queries. This attention vector is used to compute a
 10238 weighted average over the outputs of *another* convolutional neural network of the source,
 10239 yielding an averaged representation \mathbf{c}_m , which is then fed into the decoder. As with the
 10240 transformer, speed is the main advantage over recurrent encoding models; another sim-
 10241 ilarity is that word order information is approximated through the use of positional en-
 10242 codings.⁶

10243 18.3.3 Out-of-vocabulary words

10244 Thus far, we have treated translation as a problem at the level of words or phrases. For
 10245 words that do not appear in the training data, all such models will struggle. There are
 10246 two main reasons for the presence of out-of-vocabulary (OOV) words:

⁵The 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).

⁶A 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.

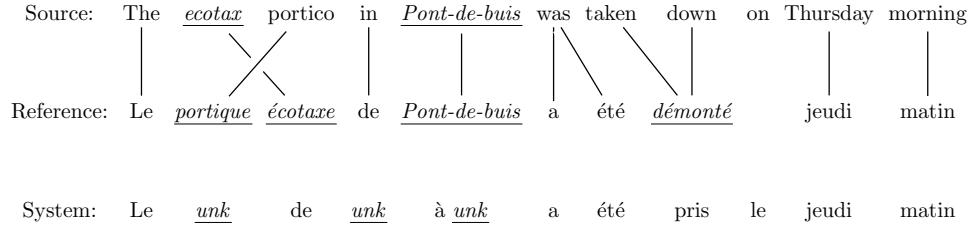


Figure 18.7: Translation with *unknown words*. The system outputs *unk* to indicate words that are outside its vocabulary. Figure adapted from Luong et al. (2015).

- 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, the vocabulary is updated, and the merging operation is applied again. For example, given the dictionary *{fish, fished, want, wanted, bike, biked}*, we would first form the subword unit *ed*, since this character bigram appears in three of the six words. Next, there are several bigrams that each appear in a pair of words: *fi, is, sh, wa, an*, etc. These can be merged in any order. By iterating this process, we eventually reach the segmentation, *{fish, fish+ed, want, want+ed, bik+e, bik+ed}*. At this point, there are no bigrams that appear more than once. In real data, merging is performed until the number of subword units reaches some predefined threshold, such as 10^4 .

10274 Each subword unit is treated as a token for translation, in both the encoder (source
 10275 side) and decoder (target side). BPE can be applied jointly to the union of the source and
 10276 target vocabularies, identifying subword units that appear in both languages. For lan-
 10277 guages that have different scripts, such as English and Russian, **transliteration** between
 10278 the scripts should be applied first.⁷

10279 **18.4 Decoding**

Given a trained translation model, the decoding task is:

$$\hat{\mathbf{w}}^{(t)} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{w}^{(s)}), \quad [18.47]$$

10280 where $\mathbf{w}^{(t)}$ is a sequence of tokens from the target vocabulary \mathcal{V} . It is not possible to
 10281 efficiently obtain exact solutions to the decoding problem, for even minimally effective
 10282 models in either statistical or neural machine translation. Today's state-of-the-art transla-
 10283 tion systems use **beam search** (see § 11.3.1), which is an incremental decoding algorithm
 10284 that maintains a small constant number of competitive hypotheses. Such greedy approxi-
 10285 mations are reasonably effective in practice, and this may be in part because the decoding
 10286 objective is only loosely correlated with measures of translation quality, so that exact op-
 10287 timization of [18.47] may not greatly improve the resulting translations.

Decoding in neural machine translation is simpler than in phrase-based statistical ma-
 chine translation.⁸ The scoring function Ψ is defined,

$$\Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}) = \sum_{m=1}^{M^{(t)}} \psi(w_m^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) \quad [18.48]$$

$$\psi(w^{(t)}; \mathbf{w}_{1:m-1}^{(t)}, \mathbf{z}) = \beta_{w_m^{(t)}} \cdot \mathbf{h}_m^{(t)} - \log \sum_{w \in \mathcal{V}} \exp(\beta_w \cdot \mathbf{h}_m^{(t)}), \quad [18.49]$$

10288 where \mathbf{z} is the encoding of the source sentence $\mathbf{w}^{(s)}$, and $\mathbf{h}_m^{(t)}$ is a function of the encoding
 10289 \mathbf{z} and the decoding history $\mathbf{w}_{1:m-1}^{(t)}$. This formulation subsumes the attentional translation
 10290 model, where \mathbf{z} is a matrix encoding of the source.

Now consider the incremental decoding algorithm,

$$\hat{w}_m^{(t)} = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \psi(w; \hat{\mathbf{w}}_{1:m-1}^{(t)}, \mathbf{z}), \quad m = 1, 2, \dots \quad [18.50]$$

⁷Transliteration 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).

⁸For more on decoding in phrase-based statistical models, see Koehn (2009).

10291 This algorithm selects the best target language word at position m , assuming that it has
 10292 already generated the sequence $\hat{w}_{1:m-1}^{(t)}$. (Termination can be handled by augmenting
 10293 the vocabulary \mathcal{V} with a special end-of-sequence token, ■.) The incremental algorithm
 10294 is likely to produce a suboptimal solution to the optimization problem defined in Equa-
 10295 tion 18.47, because selecting the highest-scoring word at position m can set the decoder
 10296 on a “garden path,” in which there are no good choices at some later position $n > m$. We
 10297 might hope for some dynamic programming solution, as in sequence labeling (§ 7.3). But
 10298 the Viterbi algorithm and its relatives rely on a Markov decomposition of the objective
 10299 function into a sum of local scores: for example, scores can consider locally adjacent tags
 10300 (y_m, y_{m-1}), but not the entire tagging history $y_{1:m}$. This decomposition is not applicable
 10301 to recurrent neural networks, because the hidden state $h_m^{(t)}$ is impacted by the entire his-
 10302 tory $w_{1:m}^{(t)}$; this sensitivity to long-range context is precisely what makes recurrent neural
 10303 networks so effective.⁹ In fact, it can be shown that decoding from any recurrent neural
 10304 network is NP-complete (Siegelmann and Sontag, 1995; Chen et al., 2018).

10305 **Beam search** Beam search is a general technique for avoiding search errors when ex-
 10306 haustive search is impossible; it was first discussed in § 11.3.1. Beam search can be seen
 10307 as a variant of the incremental decoding algorithm sketched in Equation 18.50, but at
 10308 each step m , a set of K different hypotheses are kept on the beam. For each hypothesis
 10309 $k \in \{1, 2, \dots, K\}$, 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
 10310 the current hidden state $h_k^{(t)}$. At each step in the beam search, the K top-scoring children
 10311 of each hypothesis currently on the beam are “expanded”, and the beam is updated. For
 10312 a detailed description of beam search for RNN decoding, see Graves (2012).

10313 **Learning and search** Conventionally, the learning algorithm is trained to predict the
 10314 right token in the translation, conditioned on the translation history being correct. But
 10315 if decoding must be approximate, then we might do better by modifying the learning
 10316 algorithm to be robust to errors in the translation history. **Scheduled sampling** does this
 10317 by training on histories that sometimes come from the ground truth, and sometimes come
 10318 from the model’s own output (Bengio et al., 2015).¹⁰ As training proceeds, the training
 10319 wheels come off: we increase the fraction of tokens that come from the model rather than
 10320 the ground truth. Another approach is to train on an objective that relates directly to beam
 10321 search performance (Wiseman et al., 2016). **Reinforcement learning** has also been applied
 10322 to decoding of RNN-based translation models, making it possible to directly optimize
 10323 translation metrics such as BLEU (Ranzato et al., 2016).

⁹Note 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.

¹⁰Scheduled 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.

10324 **18.5 Training towards the evaluation metric**

10325 In likelihood-based training, the objective is to maximize the probability of a parallel
 10326 corpus. However, translations are not evaluated in terms of likelihood: metrics like BLEU
 10327 consider only the correctness of a single output translation, and not the range of prob-
 10328 abilities that the model assigns. It might therefore be better to train translation models
 10329 to achieve the highest BLEU score possible — to the extent that we believe BLEU mea-
 10330 sures translation quality. Unfortunately, BLEU and related metrics are not friendly for
 10331 optimization: they are discontinuous, non-differentiable functions of the parameters of
 10332 the translation model.

Consider an error function $\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})$, which measures the discrepancy between the system translation $\hat{\mathbf{w}}^{(t)}$ and the reference translation $\mathbf{w}^{(t)}$; this function could be based on BLEU or any other metric on translation quality. One possible criterion would be to select the parameters θ that minimize the error of the system's preferred translation,

$$\hat{\mathbf{w}}^{(t)} = \operatorname{argmax}_{\mathbf{w}^{(t)}} \Psi(\mathbf{w}^{(t)}, \mathbf{w}^{(s)}; \theta) \quad [18.51]$$

$$\hat{\theta} = \operatorname{argmin}_{\theta} \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(s)}) \quad [18.52]$$

10333 However, identifying the top-scoring translation $\hat{\mathbf{w}}^{(t)}$ is usually intractable, as described
 10334 in the previous section. In **minimum error-rate training (MERT)**, $\hat{\mathbf{w}}^{(t)}$ is selected from a
 10335 set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$; this is typically a strict subset of all possible transla-
 10336 tions, so that it is only possible to optimize an approximation to the true error rate (Och
 10337 and Ney, 2003).

A further issue is that the objective function in Equation 18.52 is discontinuous and non-differentiable, due to the argmax over translations: an infinitesimal change in the parameters θ could cause another translation to be selected, with a completely different error. To address this issue, we can instead minimize the **risk**, which is defined as the expected error rate,

$$R(\theta) = E_{\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \theta} [\Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)})] \quad [18.53]$$

$$= \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} p(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}). \quad [18.54]$$

10338 **Minimum risk training** minimizes the sum of $R(\theta)$ across all instances in the training set.

The risk can be generalized by exponentiating the translation probabilities,

$$\tilde{p}(\mathbf{w}^{(t)}; \theta, \alpha) \propto \left(p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \theta) \right)^\alpha \quad [18.55]$$

$$\tilde{R}(\theta) = \sum_{\hat{\mathbf{w}}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})} \tilde{p}(\hat{\mathbf{w}}^{(t)} | \mathbf{w}^{(s)}; \alpha, \theta) \times \Delta(\hat{\mathbf{w}}^{(t)}, \mathbf{w}^{(t)}) \quad [18.56]$$

10339 where $\mathcal{Y}(\mathbf{w}^{(s)})$ is now the set of *all* possible translations for $\mathbf{w}^{(s)}$. Exponentiating the prob-
 10340 abilities in this way is known as **annealing** (Smith and Eisner, 2006). When $\alpha = 1$, then
 10341 $\tilde{R}(\boldsymbol{\theta}) = R(\boldsymbol{\theta})$; when $\alpha = \infty$, then $\tilde{R}(\boldsymbol{\theta})$ is equivalent to the sum of the errors of the maxi-
 10342 mum probability translations for each sentence in the dataset.

Clearly the set of candidate translations $\mathcal{Y}(\mathbf{w}^{(s)})$ is too large to explicitly sum over. Because the error function Δ generally does not decompose into smaller parts, there is no efficient dynamic programming solution to sum over this set. We can approximate the sum $\sum_{\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})}$ with a sum over a finite number of samples, $\{\mathbf{w}_1^{(t)}, \mathbf{w}_2^{(t)}, \dots, \mathbf{w}_K^{(t)}\}$. If these samples were drawn uniformly at random, then the (annealed) risk would be approximated as (Shen et al., 2016),

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{Z} \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha) \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}) \quad [18.57]$$

$$Z = \sum_{k=1}^K \tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta}, \alpha). \quad [18.58]$$

10343 Shen et al. (2016) report that performance plateaus at $K = 100$ for minimum risk training
 10344 of neural machine translation.

Uniform sampling over the set of all possible translations is undesirable, because most translations have very low probability. A solution from Monte Carlo estimation is **importance sampling**, in which we draw samples from a **proposal distribution** $q(\mathbf{w}^{(s)})$. This distribution can be set equal to the current translation model $p(\mathbf{w}^{(t)} | \mathbf{w}^{(s)}; \boldsymbol{\theta})$. Each sample is then weighted by an **importance score**, $\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)}; \mathbf{w}^{(s)})}$. The effect of this weighting is to correct for any mismatch between the proposal distribution q and the true distribution \tilde{p} . The risk can then be approximated as,

$$\mathbf{w}_k^{(t)} \sim q(\mathbf{w}^{(s)}) \quad [18.59]$$

$$\omega_k = \frac{\tilde{p}(\mathbf{w}_k^{(t)} | \mathbf{w}^{(s)})}{q(\mathbf{w}_k^{(t)}; \mathbf{w}^{(s)})} \quad [18.60]$$

$$\tilde{R}(\boldsymbol{\theta}) \approx \frac{1}{\sum_{k=1}^K \omega_k} \sum_{k=1}^K \omega_k \times \Delta(\mathbf{w}_k^{(t)}, \mathbf{w}^{(t)}). \quad [18.61]$$

10345 Importance sampling will generally give a more accurate approximation than uniform
 10346 sampling. The only formal requirement is that the proposal assigns non-zero probability
 10347 to every $\mathbf{w}^{(t)} \in \mathcal{Y}(\mathbf{w}^{(s)})$. For more on importance sampling and related methods, see
 10348 Robert and Casella (2013).

10349 Additional resources

10350 A complete textbook on machine translation is available from Koehn (2009). While this
10351 book precedes recent work on neural translation, a more recent draft chapter on neural
10352 translation models is also available (Koehn, 2017). Neubig (2017) provides a compre-
10353 hensive tutorial on neural machine translation, starting from first principles. The course
10354 notes from Cho (2015) are also useful. Several neural machine translation libraries are
10355 available: LAMTRAM is an implementation of neural machine translation in DYNET (Neu-
10356 big et al., 2017); OPENNMT (Klein et al., 2017) and FAIRSEQ are available in PYTORCH;
10357 TENSOR2TENSOR is an implementation of several of the Google translation models in TEN-
10358 SORFLOW (Abadi et al., 2016).

10359 Literary translation is especially challenging, even for expert human translators. Mes-
10360 sud (2014) describes some of these issues in her review of an English translation of *L'étranger*,
10361 the 1942 French novel by Albert Camus.¹¹ She compares the new translation by Sandra
10362 Smith against earlier translations by Stuart Gilbert and Matthew Ward, focusing on the
10363 difficulties presented by a single word in the first sentence:

10364 Then, too, Smith has reconsidered the book's famous opening. Camus's
10365 original is deceptively simple: "*Aujourd'hui, maman est morte.*" Gilbert influ-
10366 enced generations by offering us "Mother died today"—inscribing in Meur-
10367 sault [the narrator] from the outset a formality that could be construed as
10368 heartlessness. But *maman*, after all, is intimate and affectionate, a child's name
10369 for his mother. Matthew Ward concluded that it was essentially untranslatable
10370 ("mom" or "mummy" being not quite apt), and left it in the original French:
10371 "Maman died today." There is a clear logic in this choice; but as Smith has
10372 explained, in an interview in *The Guardian*, *maman* "didn't really tell the reader
10373 anything about the connotation." She, instead, has translated the sentence as
10374 "My mother died today."

10375 I chose "My mother" because I thought about how someone would
10376 tell another person that his mother had died. Meursault is speaking
10377 to the reader directly. "My mother died today" seemed to me the
10378 way it would work, and also implied the closeness of "maman" you
10379 get in the French.

10380 Elsewhere in the book, she has translated *maman* as "mama"—again, striving
10381 to come as close as possible to an actual, colloquial word that will carry the
10382 same connotations as *maman* does in French.

¹¹The book review is currently available online at <http://www.nybooks.com/articles/2014/06/05/camus-new-letranger/>.

10383 The passage is a reminder that while the quality of machine translation has improved
 10384 dramatically in recent years, expert human translations draw on considerations that are
 10385 beyond the ken of any contemporary computational approach.

10386 **Exercises**

10387 1. Using Google translate or another online service, translate the following example
 10388 into two different languages of your choice:

10389 (18.4) It is not down on any map; true places never are.

10390 Then translate each result back into English. Which is closer to the original? Can
 10391 you explain the differences?

10392 2. Compute the unsmoothed n -gram precisions $p_1 \dots p_4$ for the two back-translations
 10393 in the previous problem, using the original source as the reference. Your n -grams
 10394 should include punctuation, and you should segment conjunctions like *it's* into two
 10395 tokens.

10396 3. You are given the following dataset of translations from “simple” to “difficult” En-
 10397 glish:

10398 (18.5) a. *Kids like cats.*
 Children adore felines.

10399 b. *Cats hats.*
 Felines fedoras.

10400 Estimate a word-to-word statistical translation model from simple English (source)
 10401 to difficult English (target), using the expectation-maximization as described in § 18.2.2.
 10402 Compute two iterations of the algorithm by hand, starting from a uniform transla-
 10403 tion model, and using the simple alignment model $p(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}$.
 10404 Hint: in the final M-step, you will want to switch from fractions to decimals.

10405 4. Building on the previous problem, what will be the converged translation proba-
 10406 bility table? Can you state a general condition about the data, under which this
 10407 translation model will fail in the way that it fails here?

10408 5. Propose a simple alignment model that would make it possible to recover the correct
 10409 translation probabilities from the toy dataset in the previous two problems.

10410 6. Let $\ell_{m+1}^{(t)}$ represent the loss at word $m+1$ of the target, and let $h_n^{(s)}$ represent the hid-
 10411 den state at word n of the source. Write the expression for the derivative $\frac{\partial \ell_{m+1}^{(t)}}{\partial h_n^{(s)}}$ in the

sequence-to-sequence translation model expressed in Equations [18.29-18.32]. You may assume that both the encoder and decoder are one-layer LSTMs. In general, how many terms are on the shortest backpropagation path from $\ell_{m+1}^{(t)}$ to $\mathbf{h}_n^{(s)}$?

7. Now consider the neural attentional model from § 18.3.1, with sigmoid attention. The derivative $\frac{\partial \ell_{m+1}^{(t)}}{\partial z_n}$ is the sum of many paths through the computation graph; identify the shortest such path. You may assume that the initial state of the decoder recurrence $\mathbf{h}_0^{(t)}$ is *not* tied to the final state of the encoder recurrence $\mathbf{h}_{M^{(s)}}^{(s)}$.
8. Apply byte-pair encoding for the vocabulary *it*, *unit*, *unite*, until no bigram appears more than once.
9. This problem relates to the complexity of machine translation. Suppose you have an oracle that returns the list of words to include in the translation, so that your only task is to order the words. Furthermore, suppose that the scoring function over orderings is a sum over bigrams, $\sum_{m=1}^M \psi(\mathbf{w}_m^{(t)}, \mathbf{w}_{m-1}^{(t)})$. Show that the problem of finding the optimal translation is NP-complete, by reduction from a well-known problem.
10. Hand-design an attentional recurrent translation model that simply copies the input from the source to the target. You may assume an arbitrarily large hidden state, and you may assume that there is a finite maximum input length M . Specify all the weights such that the maximum probability translation of any source is the source itself. Hint: it is simplest to use a simple Elman-recurrence $\mathbf{h}_m = f(\Theta \mathbf{h}_{m-1} + \mathbf{x}_m)$ rather than an LSTM.
11. Give a synchronized derivation (§ 18.2.4) for the Spanish-English translation,

(18.6) *El pez enojado atacado.*
 The fish angry attacked.
 The angry fish attacked.

As above, the second line shows a word-for-word gloss, and the third line shows the desired translation. Use the synchronized production rule in [18.22], and design the other production rules necessary to derive this sentence pair. You may derive (*atacado*, *attacked*) directly from VP.

10440 Chapter 19

10441 Text generation

10442 In many of the most interesting problems in natural language processing, language is
10443 the output. The previous chapter described the specific case of machine translation, but
10444 there are many other applications, from summarization of research articles, to automated
10445 journalism, to dialogue systems. This chapter emphasizes three main scenarios: data-to-
10446 text, in which text is generated to explain or describe a structured record or unstructured
10447 perceptual input; text-to-text, which typically involves fusing information from multiple
10448 linguistic sources into a single coherent summary; and dialogue, in which text is generated
10449 as part of an interactive conversation with one or more human participants.

10450 19.1 Data-to-text generation

10451 In data-to-text generation, the input ranges from structured records, such as the descrip-
10452 tion of an weather forecast (as shown in Figure 19.1), to unstructured perceptual data,
10453 such as a raw image or video; the output may be a single sentence, such as an image cap-
10454 tion, or a multi-paragraph argument. Despite this diversity of conditions, all data-to-text
10455 systems share some of the same challenges (Reiter and Dale, 2000):

- 10456 • determining what parts of the data to describe;
- 10457 • planning a presentation of this information;
- 10458 • **lexicalizing** the data into words and phrases;
- 10459 • organizing words and phrases into well-formed sentences and paragraphs.

10460 The earlier stages of this process are sometimes called **content selection** and **text plan-**
10461 **ning**; the later stages are often called **surface realization**.

10462 Early systems for data-to-text generation were modular, with separate software com-
10463 ponents for each task. Artificial intelligence **planning** algorithms can be applied to both

| Temperature | | | | Cloud sky cover | |
|--------------------|------------|-------------|------------|------------------------|--------------------|
| <i>time</i> | <i>min</i> | <i>mean</i> | <i>max</i> | <i>time</i> | <i>percent (%)</i> |
| 06:00-21:00 | 9 | 15 | 21 | 06:00-09:00 | 25-50 |
| Wind speed | | | | Wind direction | |
| <i>time</i> | <i>min</i> | <i>mean</i> | <i>max</i> | <i>time</i> | <i>mode</i> |
| 06:00-21:00 | 15 | 20 | 30 | 06:00-21:00 | S |

Cloudy, with temperatures between 10 and 20 degrees. South wind around 20 mph.

Figure 19.1: An example input-output pair for the task of generating text descriptions of weather forecasts (adapted from Konstas and Lapata, 2013).

10464 the high-level information structure and the organization of individual sentences, ensur-
 10465 ing that communicative goals are met (McKeown, 1992; Moore and Paris, 1993). Surface
 10466 realization can be performed by grammars or templates, which link specific types of data
 10467 to candidate words and phrases. A simple example template is offered by Wiseman et al.
 10468 (2017), for generating descriptions of basketball games:

10469 (19.1) The <team1>(<wins1>-<losses1>) defeated the <team2>(<wins2>-<losses2>),
 10470 <pts1>-<pts2>.
 10471 The New York Knicks (45-5) defeated the Boston Celtics (11-38), 115-79.

10472 For more complex cases, it may be necessary to apply morphological inflections such as
 10473 pluralization and tense marking — even in the simple example above, languages such
 10474 as Russian would require case marking suffixes for the team names. Such inflections can
 10475 be applied as a postprocessing step. Another difficult challenge for surface realization is
 10476 the generation of varied **referring expressions** (e.g., *The Knicks, New York, they*), which is
 10477 critical to avoid repetition. As discussed in § 16.2.1, the form of referring expressions is
 10478 constrained by the discourse and information structure.

10479 An example at the intersection of rule-based and statistical techniques is the NITRO-
 10480 GEN system (Langkilde and Knight, 1998). The input to NITROGEN is an abstract meaning
 10481 representation (AMR; see § 13.3) of semantic content to be expressed in a single sentence.
 10482 In data-to-text scenarios, the abstract meaning representation is the output of a higher-
 10483 level text planning stage. A set of rules then converts the abstract meaning representation
 10484 into various sentence plans, which may differ in both the high-level structure (e.g., active
 10485 versus passive voice) as well as the low-level details (e.g., word and phrase choice). Some
 10486 examples are shown in Figure 19.2. To control the combinatorial explosion in the number
 10487 of possible realizations for any given meaning, the sentence plans are unified into a single
 10488 finite-state acceptor, in which word tokens are represented by arcs (see § 9.1.1). A bigram

```
(a / admire-01
 :ARG0 (v / visitor
       :ARG1-of (c / arrive-01
                  :ARG4 (j / Japan)))
       :ARG1 (m / "Mount Fuji"))
```

- Visitors who came to Japan admire Mount Fuji.
- Visitors who came in Japan admire Mount Fuji.
- Mount Fuji is admired by the visitor who came in Japan.

Figure 19.2: Abstract meaning representation and candidate surface realizations from the NITROGEN system. Example adapted from Langkilde and Knight (1998).

language model is then used to compute weights on the arcs, so that the shortest path is also the surface realization with the highest bigram language model probability.

More recent systems are unified models that are trained end-to-end using backpropagation. Data-to-text generation shares many properties with machine translation, including a problem of **alignment**: labeled examples provide the data and the text, but they do not specify which parts of the text correspond to which parts of the data. For example, to learn from Figure 19.1, the system must align the word *cloudy* to records in CLOUD SKY COVER, the phrases *10* and *20 degrees* to the MIN and MAX fields in TEMPERATURE, and so on. As in machine translation, both latent variables and neural attention have been proposed as solutions.

19.1.1 Latent data-to-text alignment

Given a dataset of texts and associated records $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$, our goal is to learn a model Ψ , so that

$$\hat{\mathbf{w}} = \underset{\mathbf{w} \in \mathcal{V}^*}{\operatorname{argmax}} \Psi(\mathbf{w}, \mathbf{y}; \theta), \quad [19.1]$$

where \mathcal{V}^* is the set of strings over a discrete vocabulary, and θ is a vector of parameters. The relationship between \mathbf{w} and \mathbf{y} is complex: the data \mathbf{y} may contain dozens of records, and \mathbf{w} may extend to several sentences. To facilitate learning and inference, it would be helpful to decompose the scoring function Ψ into subcomponents. This would be possible if given an **alignment**, specifying which element of \mathbf{y} is expressed in each part of \mathbf{w} . Specifically, let z_m indicates the record aligned to word m . For example, in Figure 19.1, z_1 might specify that the word *cloudy* is aligned to the record *cloud-sky-cover:percent*. The score for this alignment would then be given by the weight on features such as

$$(\textit{cloudy}, \textit{cloud-sky-cover:percent}). \quad [19.2]$$

In general, given an observed set of alignments, the score for a generation can be

10511 written as sum of local scores (Angeli et al., 2010):

$$\Psi(\mathbf{w}, \mathbf{y}; \theta) = \sum_{m=1}^M \psi_{w,y}(\mathbf{w}_m, \mathbf{y}_{z_m}) + \psi_w(w_m, w_{m-1}) + \psi_z(z_m, z_{m-1}), \quad [19.3]$$

10512 where ψ_w can represent a bigram language model, and ψ_z can be tuned to reward coherence,
 10513 such as the use of related records in nearby words.¹ The parameters of this model
 10514 could be learned from labeled data $\{(\mathbf{w}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^N$. However, while several datasets
 10515 include structured records and natural language text (Barzilay and McKeown, 2005; Chen
 10516 and Mooney, 2008; Liang and Klein, 2009), the alignments between text and records are
 10517 usually not available.² One solution is to model the problem probabilistically, treating the
 10518 alignment as a latent variable (Liang et al., 2009; Konstas and Lapata, 2013). The model
 10519 can then be estimated using expectation maximization or sampling (see chapter 5).

10520 19.1.2 Neural data-to-text generation

10521 The **encoder-decoder model** and **neural attention** were introduced in § 18.3 as methods
 10522 for neural machine translation. They can also be applied to data-to-text generation, with
 10523 the data acting as the source language (Mei et al., 2016). In neural machine translation,
 10524 the attention mechanism linked words in the source to words in the target; in data-to-
 10525 text generation, the attention mechanism can link each part of the generated text back
 10526 to a record in the data. The biggest departure from translation is in the encoder, which
 10527 depends on the form of the data.

10528 Data encoders

10529 In some types of structured records, all values are drawn from discrete sets. For example,
 10530 the birthplace of an individual is drawn from a discrete set of possible locations; the diag-
 10531 nosis and treatment of a patient are drawn from an exhaustive list of clinical codes (John-
 10532 son et al., 2016). In such cases, vector embeddings can be estimated for each field and
 10533 possible value: for example, a vector embedding for the field BIRTHPLACE, and another
 10534 embedding for the value BERKELEY_CALIFORNIA (Bordes et al., 2011). The table of such
 10535 embeddings serves as the encoding of a structured record (He et al., 2017). It is also possi-
 10536 ble to compress the entire table into a single vector representation, by **pooling** across the
 10537 embeddings of each field and value (Lebret et al., 2016).

¹More expressive decompositions of Ψ are possible. For example, Wong and Mooney (2007) use a synchronous context-free grammar (see § 18.2.4) to “translate” between a meaning representation and natural language text.

²An exception is a dataset of records and summaries from American football games, containing annotations of alignments between sentences and records (Snyder and Barzilay, 2007).

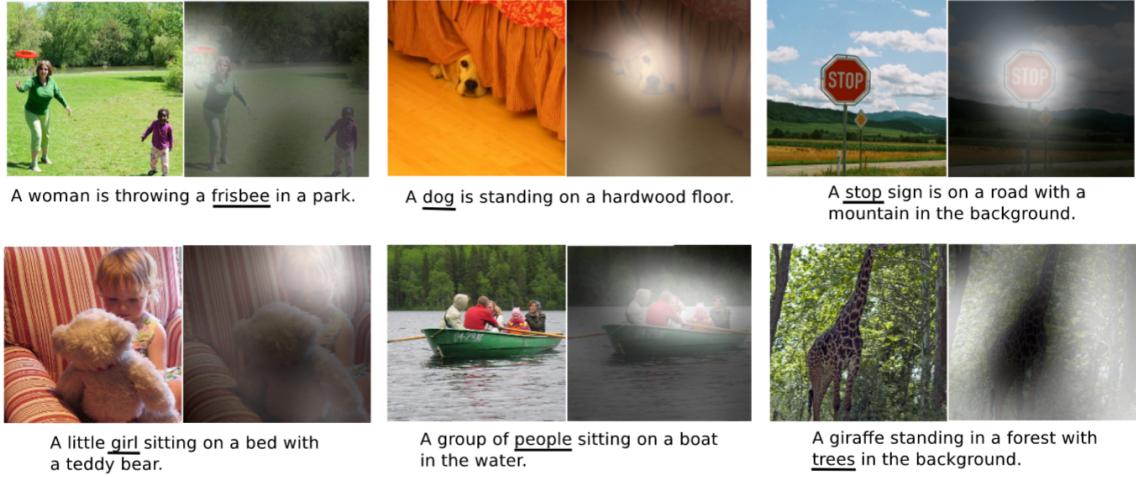


Figure 19.3: Examples of the image captioning task, with attention masks shown for each of the underlined words. From Xu et al. (2015). [todo: permission]

Sequences Some types of structured records have a natural ordering, such as events in a game (Chen and Mooney, 2008) and steps in a recipe (Tutin and Kittredge, 1992). For example, the following records describe a sequence of events in a robot soccer match (Mei et al., 2016):

```
PASS(arg1 = PURPLE6, arg2 = PURPLE3)
KICK(arg1 = PURPLE3)
BADPASS(arg1 = PURPLE3, arg2 = PINK9).
```

10538 Each event is a single record, and can be encoded by a concatenation of vector representations
 10539 for the event type (e.g., PASS), the field (e.g., arg1), and the values (e.g., PURPLE3),
 10540 e.g.,

$$\mathbf{X} = [\mathbf{u}_{\text{PASS}}, \mathbf{u}_{\text{arg1}}, \mathbf{u}_{\text{PURPLE6}}, \mathbf{u}_{\text{arg2}}, \mathbf{u}_{\text{PURPLE3}}]. \quad [19.4]$$

10541 This encoding can then act as the input layer for a recurrent neural network, yielding a
 10542 sequence of vector representations $\{\mathbf{z}_r\}_{r=1}^R$, where r indexes over records. Interestingly,
 10543 this sequence-based approach can work even in cases where there is no natural ordering
 10544 over the records, such as the weather data in Figure 19.1 (Mei et al., 2016).

10545 **Images** Another flavor of data-to-text generation is the generation of text captions for
 10546 images. Examples from this task are shown in Figure 19.3. Images are naturally repre-
 10547 sented as tensors: a color image of 320×240 pixels would be stored as a tensor with
 10548 $320 \times 240 \times 3$ intensity values. The dominant approach to image classification is to en-
 10549 code images as vectors using a combination of convolution and pooling (Krizhevsky et al.,

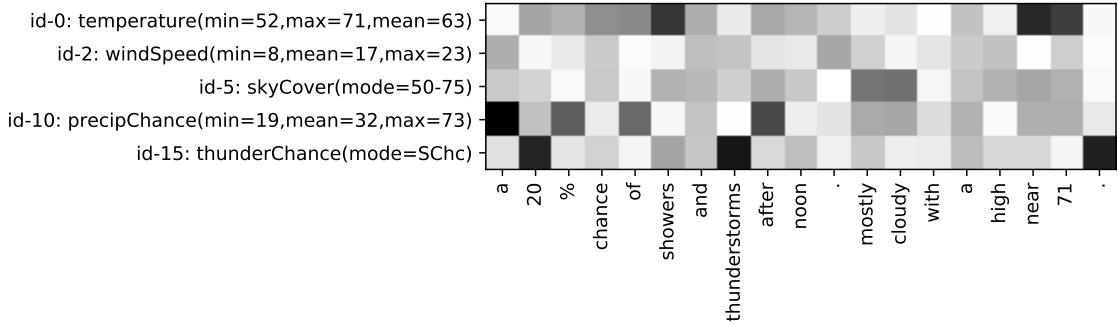


Figure 19.4: Neural attention in text generation. Figure adapted from Mei et al. (2016).

10550 2012). Chapter 3 explains how to use convolutional networks for text; for images, convolution
 10551 is applied across the vertical, horizontal, and color dimensions. By pooling the re-
 10552 sults of successive convolutions, the image is converted to a vector representation, which
 10553 can then be fed directly into the decoder as the initial state (Vinyals et al., 2015), just as
 10554 in the sequence-to-sequence translation model (see § 18.3). Alternatively, one can apply
 10555 a set of convolutional networks, yielding vector representations for different parts of the
 10556 image, which can then be combined using neural attention (Xu et al., 2015).

10557 Attention

Given a set of embeddings of the data $\{\mathbf{z}_r\}_{r=1}^R$ and a decoder state \mathbf{h}_m , an attention vector over the data can be computed using the same techniques as in machine translation (see § 18.3.1). When generating word m of the output, attention is computed over the records,

$$\psi_\alpha(m, r) = \beta_\alpha \cdot f(\Theta_\alpha[\mathbf{h}_m; \mathbf{z}_r]) \quad [19.5]$$

$$\boldsymbol{\alpha}_m = g([\psi_\alpha(m, 1), \psi_\alpha(m, 2), \dots, \psi_\alpha(m, R)]) \quad [19.6]$$

$$\mathbf{c}_m = \sum_{r=1}^R \alpha_{m \rightarrow r} \mathbf{z}_r, \quad [19.7]$$

10558 where f is an elementwise nonlinearity such as tanh or ReLU, and g is either softmax or
 10559 elementwise sigmoid. The weighted sum \mathbf{c}_m can then be included in the recurrent update
 1060 to the decoder state, or in the emission probabilities, as described in § 18.3.1. Figure 19.4
 1061 shows the attention to components of a weather record, while generating the text shown
 1062 on the x -axis.

1063 Adapting this architecture to image captioning is straightforward. A convolutional
 1064 neural networks is applied to a set of image locations, and the output at each location ℓ is
 1065 represented with a vector \mathbf{z}_ℓ . Attention can then be computed over the image locations,
 1066 as shown in the right panels of each pair of images in Figure 19.3.

10567 Various modifications to this basic mechanism have been proposed. In **coarse-to-fine**
 10568 **attention** (Mei et al., 2016), each record receives a global attention $a_r \in [0, 1]$, which is
 10569 independent of the decoder state. This global attention, which represents the overall
 10570 importance of the record, is multiplied with the decoder-based attention scores, before
 10571 computing the final normalized attentions. In **structured attention**, the attention vector
 10572 $\alpha_{m \rightarrow \cdot}$ can include structural biases, which can favor assigning higher attention values to
 10573 contiguous segments or to dependency subtrees (Kim et al., 2017). Structured attention
 10574 vectors can be computed by running the forward-backward algorithm to obtain marginal
 10575 attention probabilities (see § 7.5.3). Because each step in the forward-backward algorithm
 10576 is differentiable, it can be encoded in a computation graph, and end-to-end learning can
 10577 be performed by backpropagation.

10578 **Decoder**

10579 Given the encoding, the decoder can function just as in neural machine translation (see
 10580 § 18.3.1), using the attention-weighted encoder representation in the decoder recurrence
 10581 and/or output computation. As in machine translation, beam search can help to avoid
 10582 search errors (Lebret et al., 2016).

Many applications require generating words that do not appear in the training vocabulary. For example, a weather record may contain a previously unseen city name; a sports record may contain a previously unseen player name. Such tokens can be generated in the text by copying them over from the input (e.g., Gulcehre et al., 2016).³ First introduce an additional variable $s_m \in \{\text{gen}, \text{copy}\}$, indicating whether token $w_m^{(t)}$ should be generated or copied. The decoder probability is then,

$$p(w^{(t)} | w_{1:m-1}^{(t)}, \mathbf{Z}, s_m) = \begin{cases} \text{SoftMax}(\beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}), & s_m = \text{gen} \\ \sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}, & s_m = \text{copy}, \end{cases} \quad [19.8]$$

10583 where $\delta(w_r^{(s)} = w^{(t)})$ is an indicator function, taking the value 1 iff the text of the record
 10584 $w_r^{(s)}$ is identical to the target word $w^{(t)}$. The probability of copying record r from the source
 10585 is $\delta(s_m = \text{copy}) \times \alpha_{m \rightarrow r}$, the product of the copy probability by the local attention. Note
 10586 that in this model, the attention weights α_m are computed from the *previous* decoder state
 10587 \mathbf{h}_{m-1} . The computation graph therefore remains a feedforward network, with recurrent
 10588 paths such as $\mathbf{h}_{m-1}^{(t)} \rightarrow \alpha_m \rightarrow w_m^{(t)} \rightarrow \mathbf{h}_m^{(t)}$.

10589 To facilitate end-to-end training, the switching variable s_m can be represented by a
 10590 gate π_m , which is computed from a two-layer feedforward network, whose input consists
 10591 of the concatenation of the decoder state $\mathbf{h}_{m-1}^{(t)}$ and the attention-weighted representation

³A number of variants of this strategy have been proposed (e.g., Gu et al., 2016; Merity et al., 2017). See Wiseman et al. (2017) for an overview.

10592 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,

$$\begin{aligned} p(w^{(t)} | \mathbf{w}_{1:m}^{(t)}, \mathbf{Z}) &= \pi_m \times \underbrace{\frac{\exp \beta_{w^{(t)}} \cdot \mathbf{h}_{m-1}^{(t)}}{\sum_{j=1}^V \exp \beta_j \cdot \mathbf{h}_{m-1}^{(t)}}}_{\text{generate}} + (1 - \pi_m) \times \underbrace{\sum_{r=1}^R \delta(w_r^{(s)} = w^{(t)}) \times \alpha_{m \rightarrow r}}_{\text{copy}}. \end{aligned} \quad [19.10]$$

10593 19.2 Text-to-text generation

10594 Text-to-text generation includes problems of summarization and simplification:

- 10595 • reading a novel and outputting a paragraph-long summary of the plot;⁴
- 10596 • reading a set of blog posts about politics, and outputting a bullet list of the various issues and perspectives;
- 10598 • reading a technical research article about the long-term health consequences of drinking kombucha, and outputting a summary of the article in language that non-experts can understand.

10601 These problems can be approached in two ways: through the encoder-decoder architecture
10602 discussed in the previous section, or by operating directly on the input text.

10603 19.2.1 Neural abstractive summarization

10604 **Sentence summarization** is the task of shortening a sentence while preserving its meaning,
10605 as in the following examples (Knight and Marcu, 2000; Rush et al., 2015):

- 10606 (19.2) The documentation is typical of Epson quality: excellent.
10607 Documentation is excellent.
- 10608
- 10609 (19.3) Russian defense minister Ivanov called sunday for the creation of a joint front for
10610 combating global terrorism.
10611 Russia calls for joint front against terrorism.

⁴In § 16.3.4, 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.

10613 Sentence summarization is closely related to **sentence compression**, in which the sum-
 10614 mary is produced by deleting words or phrases from the original (Clarke and Lapata,
 10615 2008). But as shown in (19.3), a sentence summary can also introduce new words, such as
 10616 *against*, which replaces the phrase *for combatting*.

10617 Sentence summarization can be treated as a machine translation problem, using the at-
 10618 tentional encoder-decoder translation model discussed in § 18.3.1 (Rush et al., 2015). The
 10619 longer sentence is encoded into a sequence of vectors, one for each token. The decoder
 10620 then computes attention over these vectors when updating its own recurrent state. As
 10621 with data-to-text generation, it can be useful to augment the encoder-decoder model with
 10622 the ability to copy words directly from the source. Rush et al. (2015) train this model by
 10623 building four million sentence pairs from news articles. In each pair, the longer sentence is
 10624 the first sentence of the article, and the summary is the article headline. Sentence summa-
 10625 rization can also be trained in a semi-supervised fashion, using a probabilistic formulation
 10626 of the encoder-decoder model called a **variational autoencoder** (Miao and Blunsom, 2016,
 10627 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 at-
 tended 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,

$$\ell_m = \sum_{n=1}^{M^{(s)}} \min(\alpha_{m \rightarrow n}, t_{m \rightarrow n}). \quad [19.12]$$

10628 This loss will be low if α_m assigns little attention to words that already have large values in
 10629 t_m . Coverage loss is similar to the concept of **marginal relevance**, in which the reward for
 10630 adding new content is proportional to the extent to which it increases the overall amount
 10631 of information conveyed by the summary (Carbonell and Goldstein, 1998).

10632 19.2.2 Sentence fusion for multi-document summarization

10633 In **multi-document summarization**, the goal is to produce a summary that covers the
 10634 content of several documents (McKeown et al., 2002). One approach to this challenging
 10635 problem is to identify sentences across multiple documents that relate to a single theme,
 10636 and then to fuse them into a single sentence (Barzilay and McKeown, 2005). As an exam-
 10637 ple, consider the following two sentences (McKeown et al., 2010):

- 10638 (19.4) Palin actually turned against the bridge project only after it became a national
 10639 symbol of wasteful spending.
 10640 (19.5) Ms. Palin supported the bridge project while running for governor, and aban-
 10641 doned it after it became a national scandal.

10642 An *intersection* preserves only the content that is present in both sentences:

10643 (19.6) Palin turned against the bridge project after it became a national scandal.

10644 A *union* includes information from both sentences:

10645 (19.7) Ms. Palin supported the bridge project while running for governor, but turned
 10646 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),

$$\max_{\mathbf{y}} \sum_{i,j,r} \psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta}) \times y_{i,j,r} \quad [19.13]$$

$$\text{s.t. } \mathbf{y} \in \mathcal{C}, \quad [19.14]$$

10647 where the variable $y_{i,j,r} \in \{0, 1\}$ indicates whether there is an edge from i to j of type r ,
 10648 the score of this edge is $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$, and \mathcal{C} is a set of constraints, which ensures that \mathbf{y}
 10649 forms a valid dependency graph. As usual, \mathbf{w} is the list of words in the graph, and $\boldsymbol{\theta}$ is a
 10650 vector of parameters. The score $\psi(i \xrightarrow{r} j, \mathbf{w}; \boldsymbol{\theta})$ reflects the “importance” of the modifier
 10651 j to the overall meaning: in intersective fusion, this score indicates the extent to which
 10652 the content in this edge is expressed in all sentences; in union fusion, the score indicates
 10653 whether the content in the edge is expressed in any sentence. The constraint set \mathcal{C} can
 10654 impose additional linguistic constraints: for example, ensuring that coordinated nouns
 10655 are sufficiently similar. The resulting tree must then be **linearized** into a sentence. Lin-
 10656 earization is like the inverse of dependency parsing: instead of parsing from a sequence
 10657 of tokens into a tree, we must convert the tree back into a sequence of tokens. This is
 10658 typically done by generating a set of candidate linearizations, and choosing the one with
 10659 the highest score under a language model (Langkilde and Knight, 1998; Song et al., 2016).

10660 19.3 Dialogue

10661 **Dialogue systems** are capable of conversing with a human interlocutor, often to per-
 10662 form some task (Grosz, 1979), but sometimes just to chat (Weizenbaum, 1966). While re-

- (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?
 ...

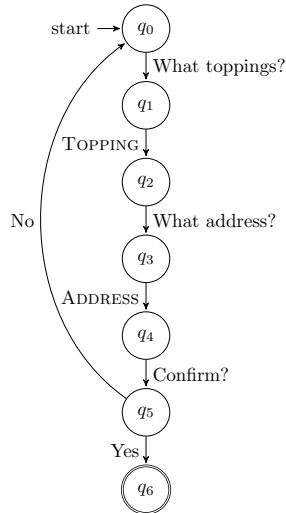


Figure 19.5: An example dialogue and the associated finite-state model. In the finite-state model, SMALL CAPS indicates that the user must provide information of this type in their answer.

10663 search on dialogue systems goes back several decades (Carbonell, 1970; Winograd, 1972),
 10664 commercial systems such as Alexa and Siri have recently brought this technology into
 10665 widespread use. Nonetheless, there is a significant gap between research and practice:
 10666 many practical dialogue systems remain scripted and inflexible, while research systems
 10667 emphasize abstractive text generation, “on-the-fly” decision making, and probabilistic
 10668 reasoning about the user’s intentions.

10669 19.3.1 Finite-state and agenda-based dialogue systems

10670 Finite-state automata were introduced in chapter 9 as a formal model of computation,
 10671 in which string inputs and outputs are linked to transitions between a finite number of
 10672 discrete states. This model naturally fits simple task-oriented dialogues, such as the one
 10673 shown in the left panel of Figure 19.5. This (somewhat frustrating) dialogue can be repre-
 10674 sented with a finite-state transducer, as shown in the right panel of the figure. The accept-
 10675 ing state is reached only when the two needed pieces of information are provided, and the
 10676 human user confirms that the order is correct. In this simple scenario, the TOPPING and
 10677 ADDRESS are the two **slots** associated with the activity of ordering a pizza, which is called
 10678 a **frame**. Frame representations can be hierarchical: for example, an ADDRESS could have
 10679 slots of its own, such as STREET and CITY.

10680 In the example dialogue in Figure 19.5, the user provides the precise inputs that are
 10681 needed in each turn (e.g., *anchovies*; *the College of Computing building*). Some users may

10682 prefer to communicate more naturally, with phrases like *I'd, uh, like some anchovies please.*
 10683 One approach to handling such utterances is to design a custom grammar, with non-
 10684 terminals for slots such as TOPPING and LOCATION. However, context-free parsing of
 10685 unconstrained speech input is challenging. A more lightweight alternative is BIO-style
 10686 sequence labeling (see § 8.3), e.g.:

10687 (19.9) *I'd like anchovies , and please bring it to the College of Computing*
 O O B-TOPPING O O O O O O B-ADDR I-ADDR I-ADDR I-ADDR
 10688 *Building .*
 I-ADDR O

10689 The tagger can be driven by a bi-directional recurrent neural network, similar to recurrent
 10690 approaches to semantic role labeling described in § 13.2.3.

10691 The input in (19.9) could not be handled by the finite-state system from Figure 19.5,
 10692 which forces the user to provide the topping first, and then the location. In this sense, the
 10693 “initiative” is driven completely by the system. **Agenda-based dialogue systems** extend
 10694 finite-state architectures by attempting to recognize all slots that are filled by the user’s re-
 10695 ply, thereby handling these more complex examples. Agenda-based systems dynamically
 10696 pose additional questions until the frame is complete (Bobrow et al., 1977; Allen et al.,
 10697 1995; Rudnicky and Xu, 1999). Such systems are said to be **mixed-initiative**, because both
 10698 the user and the system can drive the direction of the dialogue.

10699 19.3.2 Markov decision processes

10700 The task of dynamically selecting the next move in a conversation is known as **dialogue**
 10701 **management**. This problem can be framed as a **Markov decision process**, which is a
 10702 theoretical model that includes a discrete set of states, a discrete set of actions, a function
 10703 that computes the probability of transitions between states, and a function that computes
 10704 the cost or reward of action-state pairs. Let’s see how each of these elements pertains to
 10705 the pizza ordering dialogue system.

- 10706 • Each state is a tuple of information about whether the topping and address are
 10707 known, and whether the order has been confirmed. For example,

(KNOWN TOPPING, UNKNOWN ADDRESS, NOT CONFIRMED) [19.15]

10708 is a possible state. Any state in which the pizza order is confirmed is a terminal
 10709 state, and the Markov decision process stops after entering such a state.

- 10710 • The set of actions includes querying for the topping, querying for the address, and
 10711 requesting confirmation. Each action induces a probability distribution over states,
 10712 $p(s_t | a_t, s_{t-1})$. For example, requesting confirmation of the order is not likely to

10713 result in a transition to the terminal state if the topping is not yet known. This
 10714 probability distribution over state transitions may be learned from data, or it may
 10715 be specified in advance.

- 10716 • Each state-action-state tuple earns a reward, $r_a(s_t, s_{t+1})$. In the context of the pizza
 10717 ordering system, a simple reward function would be,

$$r_a(s_t, s_{t-1}) = \begin{cases} 0, & a = \text{CONFIRM}, s_t = (*, *, \text{CONFIRMED}) \\ -10, & a = \text{CONFIRM}, s_t = (*, *, \text{NOT CONFIRMED}) \\ -1, & a \neq \text{CONFIRM} \end{cases} \quad [19.16]$$

10718 This function assigns zero reward for successful transitions to the terminal state, a
 10719 large negative reward to a rejected request for confirmation, and a small negative re-
 10720 ward for every other type of action. The system is therefore rewarded for reaching
 10721 the terminal state in few steps, and penalized for prematurely requesting confirma-
 10722 tion.

10723 In a Markov decision process, a **policy** is a function $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps from states
 10724 to actions (see § 15.2.4). The value of a policy is the expected sum of discounted rewards,
 10725 $E_\pi[\sum_{t=1}^T \gamma^t r_{a_t}(s_t, s_{t+1})]$, where γ is the discount factor, $\gamma \in [0, 1)$. Discounting has the
 10726 effect of emphasizing rewards that can be obtained immediately over less certain rewards
 10727 in the distant future.

10728 An optimal policy can be obtained by dynamic programming, by iteratively updating
 10729 the **value function** $V(s)$, which is the expectation of the cumulative reward from s under
 10730 the optimal action a ,

$$V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.17]$$

10731 The value function $V(s)$ is computed in terms of $V(s')$ for all states $s' \in \mathcal{S}$. A series
 10732 of iterative updates to the value function will eventually converge to a stationary point.
 10733 This algorithm is known as **value iteration**. Given the converged value function $V(s)$, the
 10734 optimal action at each state is the argmax,

$$\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s' | s, a)[r_a(s, s') + \gamma V(s')]. \quad [19.18]$$

10735 Value iteration and related algorithms are described in detail by Sutton and Barto (1998).
 10736 For applications to dialogue systems, see Levin et al. (1998) and Walker (2000).

10737 The Markov decision process framework assumes that the current state of the dialogue
 10738 is known. In reality, the system may misinterpret the user's statements — for example,
 10739 believing that a specification of the delivery location (PEACHTREE) is in fact a specification

of the topping (PEACHES). In a **partially observable Markov decision process (POMDP)**, the system receives an *observation* o , which is probabilistically conditioned on the state, $p(o | s)$. It must therefore maintain a distribution of beliefs about which state it is in, with $q_t(s)$ indicating the degree of belief that the dialogue is in state s at time t . The POMDP formulation can help to make dialogue systems more robust to errors, particularly in the context of spoken language dialogues, where the speech itself may be misrecognized (Roy et al., 2000; Williams and Young, 2007). However, finding the optimal policy in a POMDP is computationally intractable, requiring additional approximations.

19.3.3 Neural chatbots

Chatting is a lot easier when you don't need to get anything done. **Chatbots** are systems that parry the user's input with a response that keeps the conversation going. They can be built from the encoder-decoder architecture discussed in § 18.3 and § 19.1.2: the encoder converts the user's input into a vector, and the decoder produces a sequence of words as a response. For example, Shang et al. (2015) apply the attentional encoder-decoder translation model, training on a dataset of posts and responses from the Chinese microblogging platform Sina Weibo.⁵ This approach is capable of generating replies that relate thematically to the input, as shown in the following examples:⁶

- 10757 (19.10) A: High fever attacks me every New Year's day.
10758 B: Get well soon and stay healthy!
- 10759 (19.11) A: I gain one more year. Grateful to my group, so happy.
10760 B: Getting old now. Time has no mercy.

While encoder-decoder models can generate responses that make sense in the context of the immediately preceding turn, they struggle to maintain coherence over longer 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).

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

⁵Twitter is also frequently used for construction of dialogue datasets (Ritter et al., 2011; Sordoni et al., 2015). Another source is technical support chat logs from the Ubuntu linux distribution (Uthus and Aha, 2013; Lowe et al., 2015).

⁶All examples are translated from Chinese by Shang et al. (2015).

10773 set of specialized modules: one for recognizing the user input, another for deciding what
10774 action to take, and a third for arranging the text of the system output.

10775 Recurrent neural network decoders can be integrated into Markov Decision Process
10776 dialogue systems, by conditioning the decoder on a representation of the information
10777 that is to be expressed in each turn (Wen et al., 2015). Specifically, the long short-term
10778 memory (LSTM; § 6.3) architecture is augmented so that the memory cell at turn m takes
10779 an additional input d_m , which is a representation of the slots and values to be expressed
10780 in the next turn. However, this approach still relies on additional modules to recognize
10781 the user’s utterance and to plan the overall arc of the dialogue.

10782 Another promising direction is to create embeddings for the elements in the domain:
10783 for example, the slots in a record and the entities that can fill them. The encoder then
10784 encodes not only the words of the user’s input, but the embeddings of the elements that
10785 the user mentions. Similarly, the decoder is endowed with the ability to refer to specific
10786 elements in the knowledge base. He et al. (2017) show that such a method can learn to
10787 play a collaborative dialogue game, in which both players are given a list of entities and
10788 their properties, and the goal is to find an entity that is on both players’ lists.

10789 Additional resources

10790 Gatt and Krahmer (2018) provide a comprehensive recent survey on text generation. For
10791 a book-length treatment of earlier work, see Reiter and Dale (2000). For a survey on image
10792 captioning, see Bernardi et al. (2016); for a survey of pre-neural approaches to dialogue
10793 systems, see Rieser and Lemon (2011). **Dialogue acts** were introduced in § 8.6 as a labeling
10794 scheme for human-human dialogues; they also play a critical role in task-based dialogue
10795 systems (e.g., Allen et al., 1996). The incorporation of theoretical models of dialogue into
10796 computational systems is reviewed by Jurafsky and Martin (2009, chapter 24).

10797 While this chapter has focused on the informative dimension of text generation, another
10798 line of research aims to generate text with configurable stylistic properties (Walker
10799 et al., 1997; Mairesse and Walker, 2011; Ficler and Goldberg, 2017; Hu et al., 2017). This
10800 chapter also does not address the generation of creative text such as narratives (Riedl and
10801 Young, 2010), jokes (Ritchie, 2001), poems (Colton et al., 2012), and song lyrics (Gonçalo Oliveira
10802 et al., 2007).

10803 Exercises

- 10804 1. Find an article about a professional basketball game, with an associated “box score”
10805 of statistics. Which are the first three elements in the box score that are expressed
10806 in the article? Can you identify template-based patterns that express these elements
10807 of the record? Now find a second article about a different basketball game. Does it

- mention the same first three elements of the box score? Do your templates capture how these elements are expressed in the text?
2. This exercise is to be done by a pair of students. One student should choose an article from the news or from Wikipedia, and manually perform semantic role labeling (SRL) on three short sentences or clauses. (See chapter 13 for a review of SRL.) Identify the main the semantic relation and its arguments and adjuncts. Pass this structured record — but not the original sentence — to the other student, whose job is to generate a sentence expressing the semantics. Then reverse roles, and try to regenerate three sentences from another article, based on the predicate-argument semantics.
 3. Compute the BLEU scores (see § 18.1.1) for the generated sentences in the previous problem, using the original article text as the reference.
 4. Align each token in the text of Figure 19.1 to a specific single record in the database, or to the null record \emptyset . For example, the tokens *south wind* would align to the record *wind direction: 06:00–21:00: mode=S*. How often is each token aligned to the same record as the previous token? How many transitions are there? How might a system learn to output *10 degrees* for the record *min=9*?
 5. In sentence compression and fusion, we may wish to preserve contiguous sequences of tokens (*n*-grams) and/or dependency edges. Find five short news articles with headlines. For each headline, compute the fraction of bigrams that appear in the main text of the article. Then do a manual dependency parse of the headline. For each dependency edge, count how often it appears as a dependency edge in the main text. You may use an automatic dependency parser to assist with this exercise, but check the output, and focus on UD 2.0 dependency grammar, as described in chapter 11.
 6. § 19.2.2 presents the idea of generating text from dependency trees, which requires **linearization**. Sometimes there are multiple ways that a dependency tree can be linearized. For example:
 - (19.12) The sick kids stayed at home in bed.
 - (19.13) The sick kids stayed in bed at home.
 Both sentences have an identical dependency parse: both *home* and *bed* are (oblique) dependents of *stayed*.
- Identify two more English dependency trees that can each be linearized in more than one way, and try to use a different pattern of variation in each tree. As usual, specify your trees in the Universal Dependencies 2 style, which is described in chapter 11.

7. In § 19.3.2, we considered a pizza delivery service. Let's simplify the problem to take-out, where it is only necessary to determine the topping and confirm the order. The state is a tuple in which the first element is T if the topping is specified and $?$ otherwise, and the second element is either YES or NO, depending on whether the order has been confirmed. The actions are TOPPING? (request information about the topping) and CONFIRM? (request confirmation). The state transition function is:

$$p(s_t | s_{t-1} = (?, \text{NO}), a = \text{TOPPING?}) = \begin{cases} 0.9, & s_t = (\text{T}, \text{NO}) \\ 0.1, & s_t = (?, \text{NO}). \end{cases} \quad [19.19]$$

$$p(s_t | s_{t-1} = (?, \text{NO}), a = \text{CONFIRM?}) = \begin{cases} 1, & s_t = (?, \text{NO}). \end{cases} \quad [19.20]$$

$$p(s_t | s_{t-1} = (\text{T}, \text{NO}), a = \text{TOPPING?}) = \begin{cases} 1, & s_t = (\text{T}, \text{NO}). \end{cases} \quad [19.21]$$

$$p(s_t | s_{t-1} = (\text{T}, \text{NO}), a = \text{CONFIRM?}) = \begin{cases} 0.9, & s_t = (\text{T}, \text{YES}) \\ 0.1, & s_t = (\text{T}, \text{NO}). \end{cases} \quad [19.22]$$

10843 Using the reward function defined in Equation 19.16, the discount $\gamma = 0.9$, and the
 10844 initialization $V(s) = 0$, execute three iterations of Equation 19.17. After these three
 10845 iterations, compute the optimal action in each state. You can assume that for the
 10846 terminal states, $V(*, \text{YES}) = 0$, so you only need to compute the values for non-
 10847 terminal states, $V(?, \text{NO})$ and $V(\text{T}, \text{NO})$.

- 10848 8. There are several toolkits that allow you to train encoder-decoder translation models
 10849 “out of the box”, such as FAIRSEQ (Gehring et al., 2017), xNNT (Neubig et al., 2018),
 10850 TENSOR2TENSOR (Vaswani et al., 2018), and OPENNMT (Klein et al., 2017).⁷ Use one
 10851 of these toolkits to train a chatbot dialogue system, using either the NPS dialogue
 10852 corpus that comes with NLTK (Forsyth and Martell, 2007), or, if you are feeling more
 10853 ambitious, the Ubuntu dialogue corpus (Lowe et al., 2015).

⁷<https://github.com/facebookresearch/fairseq>; <https://github.com/neulab/xnmt>;
<https://github.com/tensorflow/tensor2tensor>; <http://opennmt.net/>

10854 **Appendix A**

10855 **Probability**

10856 Probability theory provides a way to reason about random events. The sorts of random
10857 events that are typically used to explain probability theory include coin flips, card draws,
10858 and the weather. It may seem odd to think about the choice of a word as akin to the flip of
10859 a coin, particularly if you are the type of person to choose words carefully. But random or
10860 not, language has proven to be extremely difficult to model deterministically. Probability
10861 offers a powerful tool for modeling and manipulating linguistic data.

10862 Probability can be thought of in terms of **random outcomes**: for example, a single coin
10863 flip has two possible outcomes, heads or tails. The set of possible outcomes is the **sample**
10864 **space**, and a subset of the **sample space** is an **event**. For a sequence of two coin flips,
10865 there are four possible outcomes, $\{HH, HT, TH, TT\}$, representing the ordered sequences
10866 heads-head, heads-tails, tails-heads, and tails-tails. The event of getting exactly one head
10867 includes two outcomes: $\{HT, TH\}$.

10868 Formally, a probability is a function from events to the interval between zero and one:
10869 $\Pr : \mathcal{F} \rightarrow [0, 1]$, where \mathcal{F} is the set of possible events. An event that is certain has proba-
10870 bility one; an event that is impossible has probability zero. For example, the probability
10871 of getting fewer than three heads on two coin flips is one. Each outcome is also an event
10872 (a set with exactly one element), and for two flips of a fair coin, the probability of each
10873 outcome is,

$$\Pr(\{HH\}) = \Pr(\{HT\}) = \Pr(\{TH\}) = \Pr(\{TT\}) = \frac{1}{4}. \quad [\text{A.1}]$$

10874 **A.1 Probabilities of event combinations**

10875 Because events are sets of outcomes, we can use set-theoretic operations such as comple-
10876 ment, intersection, and union to reason about the probabilities of events and their combi-
10877 nations.

10878 For any event A , there is a **complement** $\neg A$, such that:

- 10879 • The probability of the union $A \cup \neg A$ is $\Pr(A \cup \neg A) = 1$;
- 10880 • The intersection $A \cap \neg A = \emptyset$ is the empty set, and $\Pr(A \cap \neg A) = 0$.

10881 In the coin flip example, the event of obtaining a single head on two flips corresponds to
 10882 the set of outcomes $\{HT, TH\}$; the complement event includes the other two outcomes,
 10883 $\{TT, HH\}$.

10884 A.1.1 Probabilities of disjoint events

10885 When two events have an empty intersection, $A \cap B = \emptyset$, they are **disjoint**. The probabili-
 10886 ty of the union of two disjoint events is equal to the sum of their probabilities,

$$A \cap B = \emptyset \Rightarrow \Pr(A \cup B) = \Pr(A) + \Pr(B). \quad [A.2]$$

10887 This is the **third axiom of probability**, and it can be generalized to any countable sequence
 10888 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,

$$\Pr(\{HT, TH\}) = \Pr(\{HT\} \cup \{TH\}) = \Pr(\{HT\}) + \Pr(\{TH\}) \quad [A.3]$$

$$= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \quad [A.4]$$

10889 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:

$$\Pr(B - (A \cap B)) = \Pr(B) - \Pr(A \cap B) \quad [A.8]$$

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B). \quad [A.9]$$

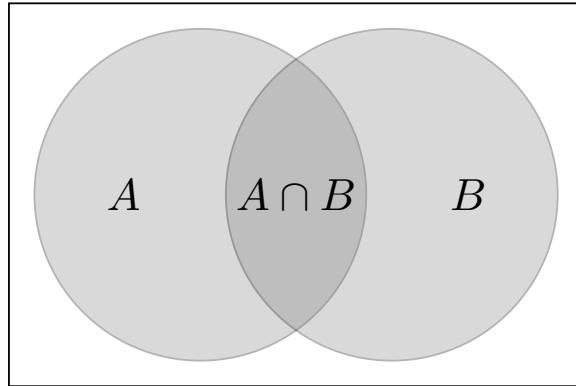


Figure A.1: A visualization of the probability of non-disjoint events A and B .

10890 A.1.2 Law of total probability

10891 A set of events $\mathcal{B} = \{B_1, B_2, \dots, B_N\}$ is a **partition** of the sample space iff each pair of
 10892 events is disjoint ($B_i \cap B_j = \emptyset$), and the union of the events is the entire sample space.
 10893 The law of total probability states that we can **marginalize** over these events as follows,

$$\Pr(A) = \sum_{B_n \in \mathcal{B}} \Pr(A \cap B_n). \quad [\text{A.10}]$$

10894 For any event B , the union $B \cup \neg B$ is a partition of the sample space. Therefore, a special
 10895 case of the law of total probability is,

$$\Pr(A) = \Pr(A \cap B) + \Pr(A \cap \neg B). \quad [\text{A.11}]$$

10896 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:

$$\Pr(A | B) = \frac{\Pr(A \cap B)}{\Pr(B)} = \frac{\Pr(B | A) \times \Pr(A)}{\Pr(B)} \quad [\text{A.13}]$$

10897 Each term in Bayes rule has a name, which we will occasionally use:

- 10898 • Pr(A) is the **prior**, since it is the probability of event A without knowledge about
10899 whether B happens or not.
- 10900 • Pr($B | A$) is the **likelihood**, the probability of event B given that event A has oc-
10901 curred.
- 10902 • Pr($A | B$) is the **posterior**, the probability of event A with knowledge that B has
10903 occurred.

10904 **Example** The classic examples for Bayes' rule involve tests for rare diseases, but Man-
10905 ning and Schütze (1999) reframe this example in a linguistic setting. Suppose that you are
10906 interested in a rare syntactic construction, such as *parasitic gaps*, which occur on average
10907 once in 100,000 sentences. Here is an example of a parasitic gap:

10908 (A.1) *Which class did you attend ... without registering for ...?*

10909 Lana Linguist has developed a complicated pattern matcher that attempts to identify
10910 sentences with parasitic gaps. It's pretty good, but it's not perfect:

- 10911 • If a sentence has a parasitic gap, the pattern matcher will find it with probability
10912 0.95. (This is the **recall**, which is one minus the **false negative rate**.)
- 10913 • If the sentence doesn't have a parasitic gap, the pattern matcher will wrongly say it
10914 does with probability 0.005. (This is the **false positive rate**, which is one minus the
10915 **precision**.)

10916 Suppose that Lana's pattern matcher says that a sentence contains a parasitic gap. What
10917 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:

$$\Pr(G | T) = \frac{\Pr(T | G) \times \Pr(G)}{\Pr(T)}. \quad [\text{A.14}]$$

10918 We already know both terms in the numerator: $\Pr(T | G)$ is the recall, which is 0.95; $\Pr(G)$
10919 is the prior, which is 10^{-5} .

10920 We are not given the denominator, but it can be computed using tools developed earlier
10921 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,

$$\Pr(G | T) = \frac{\Pr(T | G) \Pr(G)}{\Pr(T)} \quad [\text{A.20}]$$

$$= \frac{0.95 \times 10^{-5}}{0.95 \times 10^{-5} + 0.005 \times (1 - 10^{-5})} \quad [\text{A.21}]$$

$$\approx 0.002. \quad [\text{A.22}]$$

10922 Lana's pattern matcher seems accurate, with false positive and false negative rates
10923 below 5%. Yet the extreme rarity of the phenomenon means that a positive result from the
10924 detector is most likely to be wrong.

10925 A.3 Independence

Two events are independent if the probability of their intersection is equal to the product of their probabilities: $\Pr(A \cap B) = \Pr(A) \times \Pr(B)$. For example, for two flips of a fair

coin, the probability of getting heads on the first flip is independent of the probability of getting heads on the second flip:

$$\Pr(\{HT, HH\}) = \Pr(HT) + \Pr(HH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.23]$$

$$\Pr(\{HH, TH\}) = \Pr(HH) + \Pr(TH) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \quad [A.24]$$

$$\Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \quad [A.25]$$

$$\Pr(\{HT, HH\} \cap \{HH, TH\}) = \Pr(HH) = \frac{1}{4} \quad [A.26]$$

$$= \Pr(\{HT, HH\}) \times \Pr(\{HH, TH\}). \quad [A.27]$$

If $\Pr(A \cap B \mid C) = \Pr(A \mid C) \times \Pr(B \mid C)$, then the events A and B are **conditionally independent**, written $A \perp B \mid C$. Conditional independence plays a important role in probabilistic models such as Naïve Bayes chapter 2.

A.4 Random variables

Random variables are functions from events to \mathbb{R}^n , where \mathbb{R} is the set of real numbers. This subsumes several useful special cases:

- An **indicator random variable** is a function from events to the set $\{0, 1\}$. In the coin flip example, we can define Y as an indicator random variable, taking the value 1 when the coin has come up heads on at least one flip. This would include the outcomes $\{HH, HT, TH\}$. The probability $\Pr(Y = 1)$ is the sum of the probabilities of these outcomes, $\Pr(Y = 1) = \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = \frac{3}{4}$.
- A **discrete random variable** is a function from events to a discrete subset of \mathbb{R} . Consider the coin flip example: the number of heads on two flips, X , can be viewed as a discrete random variable, $X \in \{0, 1, 2\}$. The event probability $\Pr(X = 1)$ can again be computed as the sum of the probabilities of the events in which there is one head, $\{HT, TH\}$, giving $\Pr(X = 1) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

Each possible value of a random variable is associated with a subset of the sample space. In the coin flip example, $X = 0$ is associated with the event $\{TT\}$, $X = 1$ is associated with the event $\{HT, TH\}$, and $X = 2$ is associated with the event $\{HH\}$. Assuming a fair coin, the probabilities of these events are, respectively, $1/4$, $1/2$, and $1/4$. This list of numbers represents the **probability distribution** over X , written p_X , which maps from the possible values of X to the non-negative reals. For a specific value x , we write $p_X(x)$, which is equal to the event probability $\Pr(X = x)$.¹ The function p_X is called

¹In general, capital letters (e.g., X) refer to random variables, and lower-case letters (e.g., x) refer to specific values. When the distribution is clear from context, I will simply write $p(x)$.

a probability **mass** function (pmf) if X is discrete; it is called a probability **density** function (pdf) if X is continuous. In either case, the function must sum to one, and all values must be non-negative:

$$\int_x p_X(x)dx = 1 \quad [A.28]$$

$$\forall x, p_X(x) \geq 0. \quad [A.29]$$

Probabilities over multiple random variables can written as **joint probabilities**, e.g., $p_{A,B}(a,b) = \Pr(A = a \cap B = b)$. Several properties of event probabilities carry over to probability distributions over random variables:

- The **marginal probability distribution** is $p_A(a) = \sum_b p_{A,B}(a,b)$.
- The **conditional probability distribution** is $p_{A|B}(a | b) = \frac{p_{A,B}(a,b)}{p_B(b)}$.
- Random variables A and B are independent iff $p_{A,B}(a,b) = p_A(a) \times p_B(b)$.

A.5 Expectations

Sometimes we want the **expectation** of a function, such as $E[g(x)] = \sum_{x \in \mathcal{X}} g(x)p(x)$. Expectations are easiest to think about in terms of probability distributions over discrete events:

- If it is sunny, Lucia will eat three ice creams.
- If it is rainy, she will eat only one ice cream.
- There's a 80% chance it will be sunny.
- The expected number of ice creams she will eat is $0.8 \times 3 + 0.2 \times 1 = 2.6$.

If the random variable X is continuous, the expectation is an integral:

$$E[g(x)] = \int_{\mathcal{X}} g(x)p(x)dx \quad [A.30]$$

For example, a fast food restaurant in Quebec has a special offer for cold days: they give a 1% discount on poutine for every degree below zero. Assuming a thermometer with infinite precision, the expected price would be an integral over all possible temperatures,

$$E[\text{price}(x)] = \int_{\mathcal{X}} \min(1, 1+x) \times \text{original-price} \times p(x)dx. \quad [A.31]$$

10960 **A.6 Modeling and estimation**

10961 **Probabilistic models** provide a principled way to reason about random events and ran-
10962 dom variables. Let's consider the coin toss example. Each toss can be modeled as a ran-
10963 dom event, with probability θ of the event H , and probability $1 - \theta$ of the complementary
10964 event T . If we write a random variable X as the total number of heads on three coin
10965 flips, then the distribution of X depends on θ . In this case, X is distributed as a **binomial**
10966 **random variable**, meaning that it is drawn from a binomial distribution, with **parameters**
10967 $(\theta, N = 3)$. This is written,

$$X \sim \text{Binomial}(\theta, N = 3). \quad [\text{A.32}]$$

10968 The properties of the binomial distribution enable us to make statements about the X ,
10969 such as its expected value and the likelihood that its value will fall within some interval.

Now suppose that θ is unknown, but we have run an experiment, in which we ex-
 ecuted N trials, and obtained x heads. We can **estimate** θ by the principle of **maximum**
likelihood:

$$\hat{\theta} = \operatorname{argmax}_{\theta} p_X(x; \theta, N). \quad [\text{A.33}]$$

This says that the estimate $\hat{\theta}$ should be the value that maximizes the likelihood of the
 data. The semicolon indicates that θ and N are parameters of the probability function.
 The likelihood $p_X(x; \theta, N)$ can be computed from the binomial distribution,

$$p_X(x; \theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1-\theta)^{N-x}. \quad [\text{A.34}]$$

10970 This likelihood is proportional to the product of the probability of individual out-
10971 comes: for example, the sequence T, H, H, T, H would have probability $\theta^3(1-\theta)^2$. The
10972 term $\frac{N!}{x!(N-x)!}$ arises from the many possible orderings by which we could obtain x heads
10973 on N trials. This term does not depend on θ , so it can be ignored during estimation.

In practice, we maximize the log-likelihood, which is a monotonic function of the like-
 lihood. Under the binomial distribution, the log-likelihood is a **convex** function of θ (see

§ 2.4), so it can be maximized by taking the derivative and setting it equal to zero.

$$\ell(\theta) = x \log \theta + (N - x) \log(1 - \theta) \quad [\text{A.35}]$$

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{x}{\theta} - \frac{N - x}{1 - \theta} \quad [\text{A.36}]$$

$$\frac{N - x}{1 - \theta} = \frac{x}{\theta} \quad [\text{A.37}]$$

$$\frac{N - x}{x} = \frac{1 - \theta}{\theta} \quad [\text{A.38}]$$

$$\frac{N}{x} - 1 = \frac{1}{\theta} - 1 \quad [\text{A.39}]$$

$$\hat{\theta} = \frac{x}{N}. \quad [\text{A.40}]$$

10974 In this case, the maximum likelihood estimate is equal to $\frac{x}{N}$, the fraction of trials that
 10975 came up heads. This intuitive solution is also known as the **relative frequency estimate**,
 10976 since it is equal to the relative frequency of the outcome.

Is maximum likelihood estimation always the right choice? Suppose you conduct one trial, and get heads. Would you conclude that $\theta = 1$, meaning that the coin is guaranteed to come up heads? If not, then you must have some **prior expectation** about θ . To incorporate this prior information, we can treat θ as a random variable, and use Bayes' rule:

$$p(\theta | x; N) = \frac{p(x | \theta) \times p(\theta)}{p(x)} \quad [\text{A.41}]$$

$$\propto p(x | \theta) \times p(\theta) \quad [\text{A.42}]$$

$$\hat{\theta} = \operatorname{argmax}_{\theta} p(x | \theta) \times p(\theta). \quad [\text{A.43}]$$

10977 This is the **maximum a posteriori** (MAP) estimate. Given a form for $p(\theta)$, you can de-
 10978 rive the MAP estimate using the same approach that was used to derive the maximum
 10979 likelihood estimate.

10980 Additional resources

10981 A good introduction to probability theory is offered by Manning and Schütze (1999),
 10982 which helped to motivate this section. For more detail, Sharon Goldwater provides an-
 10983 other useful reference, <http://homepages.inf.ed.ac.uk/sgwater/teaching/general/probability.pdf>. A historical and philosophical perspective on probability is offered
 10984 by Diaconis and Skyrms (2017).

10986 **Appendix B**

10987 **Numerical optimization**

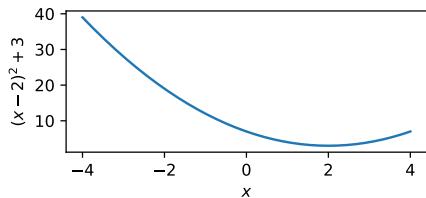
10988 Unconstrained numerical optimization involves solving problems of the form,

$$\min_{\mathbf{x} \in \mathbb{R}^D} f(\mathbf{x}), \quad [\text{B.1}]$$

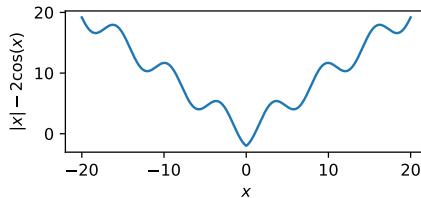
10989 where $\mathbf{x} \in \mathbb{R}^D$ is a vector of D real numbers.

10990 Differentiation is fundamental to numerical optimization. Suppose that at some \mathbf{x}^* ,
10991 every partial derivative of f is equal to 0: formally, $\frac{\partial f}{\partial x_i} \Big|_{\mathbf{x}^*} = 0$. Then \mathbf{x}^* is said to be a
10992 **critical point** of f . If f is a **convex** function (defined in § 2.4), then the value of $f(\mathbf{x}^*)$ is
10993 equal to the global minimum 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,

$$\frac{\partial d}{\partial x_1} = x_1 - 2 \quad [B.2]$$

$$\frac{\partial d}{\partial x_2} = x_2 - 1 \quad [B.3]$$

10994 The unique minimum is $\mathbf{x}^* = [2, 1]^\top$.

10995 For non-convex functions, critical points are not necessarily global minima. A **local**
 10996 **minimum** \mathbf{x}^* is a point at which the function takes a smaller value than at all nearby
 10997 neighbors: formally, \mathbf{x}^* is a local minimum if there is some positive ϵ such that $f(\mathbf{x}^*) \leq$
 10998 $f(\mathbf{x})$ for all \mathbf{x} within distance ϵ of \mathbf{x}^* . Figure B.1b shows the function $f(x) = |x| - 2 \cos(x)$,
 10999 which has many local minima, as well as a unique global minimum at $x = 0$. A critical
 11000 point may also be the local or global maximum of the function; it may be a **saddle point**,
 11001 which is a minimum with respect to at least one coordinate, and a maximum with respect
 11002 to at least one other coordinate; it may be an **inflection point**, which is neither a minimum
 11003 nor maximum. When available, the second derivative of f can help to distinguish these
 11004 cases.

11005 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,

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} - \eta^{(t)} \nabla_{\mathbf{x}^{(t)}} f. \quad [B.4]$$

11006 where $\eta^{(t)} > 0$ is a **step size**. If the step size is chosen appropriately, this procedure will
 11007 find the global minimum of a differentiable convex function. For non-convex functions,
 11008 gradient descent will find a local minimum. The extension to non-differentiable convex
 11009 functions is discussed in § 2.4.

11010 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 $g_c(\mathbf{x})$ is a scalar function of \mathbf{x} . For example, suppose that \mathbf{x} must be non-negative, and that its sum cannot exceed a budget b . Then there are $D + 1$ inequality constraints,

$$g_i(\mathbf{x}) = -x_i, \quad \forall i = 1, 2, \dots, D \quad [\text{B.7}]$$

$$g_{D+1}(\mathbf{x}) = -b + \sum_{i=1}^D x_i. \quad [\text{B.8}]$$

Inequality constraints can be combined with the original objective function f by forming a **Lagrangian**,

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{c=1}^C \lambda_c g_c(\mathbf{x}), \quad [\text{B.9}]$$

where λ_c is a **Lagrange multiplier**. For any Lagrangian, there is a corresponding dual form, which is a function of $\boldsymbol{\lambda}$:

$$D(\boldsymbol{\lambda}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}). \quad [\text{B.10}]$$

The Lagrangian L can be referred to as the **primal form**.

B.3 Example: Passive-aggressive online learning

Sometimes it is possible to solve a constrained optimization problem by manipulating the Lagrangian. One example is maximum-likelihood estimation of a Naïve Bayes probability model, as described in § 2.2.3. In that case, it is unnecessary to explicitly compute the Lagrange multiplier. Another example is illustrated by the **passive-aggressive** algorithm for online learning (Crammer et al., 2006). This algorithm is similar to the perceptron, but the goal at each step is to make the most conservative update that gives zero margin loss on the current example.¹ Each update can be formulated as a constrained optimization over the weights $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|^2 \quad [\text{B.11}]$$

$$\text{s.t. } \ell^{(i)}(\boldsymbol{\theta}) = 0 \quad [\text{B.12}]$$

where $\boldsymbol{\theta}^{(i-1)}$ is the previous set of weights, and $\ell^{(i)}(\boldsymbol{\theta})$ is the margin loss on instance i . As in § 2.4.1, this loss is defined as,

$$\ell^{(i)}(\boldsymbol{\theta}) = 1 - \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y^{(i)}) + \max_{y \neq y^{(i)}} \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}^{(i)}, y). \quad [\text{B.13}]$$

¹This is the basis for the name of the algorithm: it is passive when the loss is zero, but it aggressively moves to make the loss zero when necessary.

When the margin loss is zero for $\theta^{(i-1)}$, the optimal solution is $\theta^* = \theta^{(i-1)}$, so we will focus on the case where $\ell^{(i)}(\theta^{(i-1)}) > 0$. The Lagrangian for this problem is,

$$L(\theta, \lambda) = \frac{1}{2} \|\theta - \theta^{(i-1)}\|^2 + \lambda \ell^{(i)}(\theta), \quad [\text{B.14}]$$

Holding λ constant, we can solve for θ by differentiating,

$$\nabla_{\theta} L = \theta - \theta^{(i-1)} + \lambda \frac{\partial}{\partial \theta} \ell^{(i)}(\theta) \quad [\text{B.15}]$$

$$\theta^* = \theta^{(i-1)} + \lambda \delta, \quad [\text{B.16}]$$

where $\delta = f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})$ and $\hat{y} = \operatorname{argmax}_{y \neq y^{(i)}} \theta \cdot f(x^{(i)}, y)$.

The Lagrange multiplier λ acts as the learning rate in a perceptron-style update to θ . We can solve for λ by plugging θ^* back into the Lagrangian, obtaining the dual function,

$$D(\lambda) = \frac{1}{2} \|\theta^{(i-1)} + \lambda \delta - \theta^{(i-1)}\|^2 + \lambda(1 - (\theta^{(i-1)} + \lambda \delta) \cdot \delta) \quad [\text{B.17}]$$

$$= \frac{\lambda^2}{2} \|\delta\|^2 - \lambda^2 \|\delta\|^2 + \lambda(1 - \theta^{(i-1)} \cdot \delta) \quad [\text{B.18}]$$

$$= -\frac{\lambda^2}{2} \|\delta\|^2 + \lambda \ell^{(i)}(\theta^{(i-1)}). \quad [\text{B.19}]$$

Differentiating and solving for λ ,

$$\frac{\partial D}{\partial \lambda} = -\lambda \|\delta\|^2 + \ell^{(i)}(\theta^{(i-1)}) \quad [\text{B.20}]$$

$$\lambda^* = \frac{\ell^{(i)}(\theta^{(i-1)})}{\|\delta\|^2}. \quad [\text{B.21}]$$

The complete update equation is therefore:

$$\theta^* = \theta^{(i-1)} + \frac{\ell^{(i)}(\theta^{(i-1)})}{\|f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})\|^2} (f(x^{(i)}, y^{(i)}) - f(x^{(i)}, \hat{y})). \quad [\text{B.22}]$$

This learning rate makes intuitive sense. The numerator 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.

11027

Bibliography

- 11028 Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis,
11029 J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia,
11030 R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore,
11031 D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A.
11032 Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Watten-
11033 berg, M. Wicke, Y. Yu, and X. Zheng (2016). Tensorflow: Large-scale machine learning
11034 on heterogeneous distributed systems. *CoRR abs/1603.04467*.
- 11035 Abend, O. and A. Rappoport (2017). The state of the art in semantic representation. In
11036 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11037 Abney, S., R. E. Schapire, and Y. Singer (1999). Boosting applied to tagging and PP attach-
11038 ment. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
11039 132–134.
- 11040 Abney, S. P. (1987). *The English noun phrase in its sentential aspect*. Ph. D. thesis, Mas-
11041 sachusetts Institute of Technology.
- 11042 Abney, S. P. and M. Johnson (1991). Memory requirements and local ambiguities of pars-
11043 ing strategies. *Journal of Psycholinguistic Research* 20(3), 233–250.
- 11044 Adafre, S. F. and M. De Rijke (2006). Finding similar sentences across multiple languages
11045 in wikipedia. In *Proceedings of the Workshop on NEW TEXT Wikis and blogs and other*
11046 *dynamic text sources*.
- 11047 Ahn, D. (2006). The stages of event extraction. In *Proceedings of the Workshop on Annotating*
11048 *and Reasoning about Time and Events*, pp. 1–8. Association for Computational Linguistics.
- 11049 Aho, A. V., M. S. Lam, R. Sethi, and J. D. Ullman (2006). Compilers: Principles, techniques,
11050 & tools.
- 11051 Aikhenvald, A. Y. (2004). *Evidentiality*. Oxford University Press.

- 11052 Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on
11053 Automatic Control* 19(6), 716–723.
- 11054 Akmajian, A., R. A. Demers, A. K. Farmer, and R. M. Harnish (2010). *Linguistics: An
11055 introduction to language and communication* (Sixth ed.). Cambridge, MA: MIT press.
- 11056 Alfano, M., D. Hovy, M. Mitchell, and M. Strube (2018). Proceedings of the second acl
11057 workshop on ethics in natural language processing. In *Proceedings of the Second ACL
11058 Workshop on Ethics in Natural Language Processing*. Association for Computational Lin-
11059 guistics.
- 11060 Alfau, F. (1999). *Chromos*. Dalkey Archive Press.
- 11061 Allauzen, C., M. Riley, J. Schalkwyk, W. Skut, and M. Mohri (2007). OpenFst: A gen-
11062 eral and efficient weighted finite-state transducer library. In *International Conference on
11063 Implementation and Application of Automata*, pp. 11–23. Springer.
- 11064 Allen, J. F. (1984). Towards a general theory of action and time. *Artificial intelligence* 23(2),
11065 123–154.
- 11066 Allen, J. F., B. W. Miller, E. K. Ringger, and T. Sikorski (1996). A robust system for natural
11067 spoken dialogue. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11068 62–70.
- 11069 Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light,
11070 N. Martin, B. Miller, M. Poesio, and D. Traum (1995). The TRAINS project: A case
11071 study in building a conversational planning agent. *Journal of Experimental & Theoretical
11072 Artificial Intelligence* 7(1), 7–48.
- 11073 Alm, C. O., D. Roth, and R. Sproat (2005). Emotions from text: machine learning for
11074 text-based emotion prediction. In *Proceedings of Empirical Methods for Natural Language
11075 Processing (EMNLP)*, pp. 579–586.
- 11076 Aluísio, S., J. Pelizzoni, A. Marchi, L. de Oliveira, R. Manenti, and V. Marquiafável (2003).
11077 An account of the challenge of tagging a reference corpus for Brazilian Portuguese.
11078 *Computational Processing of the Portuguese Language*, 194–194.
- 11079 Anand, P., M. Walker, R. Abbott, J. E. Fox Tree, R. Bowman, and M. Minor (2011). Cats rule
11080 and dogs drool!: Classifying stance in online debate. In *Proceedings of the 2nd Workshop
11081 on Computational Approaches to Subjectivity and Sentiment Analysis*, Portland, Oregon, pp.
11082 1–9. Association for Computational Linguistics.
- 11083 Anandkumar, A. and R. Ge (2016). Efficient approaches for escaping higher order saddle
11084 points in non-convex optimization. In *Proceedings of the Conference On Learning Theory
11085 (COLT)*, pp. 81–102.

- 11086 Anandkumar, A., R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky (2014). Tensor decompo-
11087 sitions for learning latent variable models. *The Journal of Machine Learning Research* 15(1),
11088 2773–2832.
- 11089 Ando, R. K. and T. Zhang (2005). A framework for learning predictive structures from
11090 multiple tasks and unlabeled data. *The Journal of Machine Learning Research* 6, 1817–
11091 1853.
- 11092 Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and
11093 M. Collins (2016). Globally normalized transition-based neural networks. In *Proceedings*
11094 of the Association for Computational Linguistics (ACL), pp. 2442–2452.
- 11095 Angeli, G., P. Liang, and D. Klein (2010). A simple domain-independent probabilistic ap-
11096 proach to generation. In *Proceedings of Empirical Methods for Natural Language Processing*
11097 (EMNLP), pp. 502–512.
- 11098 Antol, S., A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh
11099 (2015). Vqa: Visual question answering. In *Proceedings of the International Conference on*
11100 *Computer Vision (ICCV)*, pp. 2425–2433.
- 11101 Aronoff, M. (1976). *Word formation in generative grammar*. MIT Press.
- 11102 Arora, S. and B. Barak (2009). *Computational complexity: a modern approach*. Cambridge
11103 University Press.
- 11104 Arora, S., R. Ge, Y. Halpern, D. Mimno, A. Moitra, D. Sontag, Y. Wu, and M. Zhu (2013).
11105 A practical algorithm for topic modeling with provable guarantees. In *Proceedings of the*
11106 *International Conference on Machine Learning (ICML)*, pp. 280–288.
- 11107 Arora, S., Y. Li, Y. Liang, T. Ma, and A. Risteski (2016). Linear algebraic structure of word
11108 senses, with applications to polysemy. *arXiv preprint arXiv:1601.03764*.
- 11109 Artstein, R. and M. Poesio (2008). Inter-coder agreement for computational linguistics.
11110 *Computational Linguistics* 34(4), 555–596.
- 11111 Artzi, Y. and L. Zettlemoyer (2013). Weakly supervised learning of semantic parsers for
11112 mapping instructions to actions. *Transactions of the Association for Computational Linguis-*
11113 *tics* 1, 49–62.
- 11114 Attardi, G. (2006). Experiments with a multilanguage non-projective dependency parser.
11115 In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 166–170.
- 11116 Auer, P. (2013). *Code-switching in conversation: Language, interaction and identity*. Routledge.
- 11117 Auer, S., C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives (2007). Dbpedia: A
11118 nucleus for a web of open data. *The semantic web*, 722–735.

- 11119 Austin, J. L. (1962). *How to do things with words*. Oxford University Press.
- 11120 Aw, A., M. Zhang, J. Xiao, and J. Su (2006). A phrase-based statistical model for SMS text
11121 normalization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11122 33–40.
- 11123 Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer normalization. *arXiv preprint
arXiv:1607.06450*.
- 11125 Bagga, A. and B. Baldwin (1998a). Algorithms for scoring coreference chains. In *Proceed-
11126 ings of the Language Resources and Evaluation Conference*, pp. 563–566.
- 11127 Bagga, A. and B. Baldwin (1998b). Entity-based cross-document coreferencing using the
11128 vector space model. In *Proceedings of the International Conference on Computational Lin-
11129 guistics (COLING)*, pp. 79–85.
- 11130 Bahdanau, D., K. Cho, and Y. Bengio (2014). Neural machine translation by jointly learn-
11131 ing to align and translate. In *Neural Information Processing Systems (NIPS)*.
- 11132 Baldwin, T. and S. N. Kim (2010). Multiword expressions. In *Handbook of natural language
11133 processing*, Volume 2, pp. 267–292. Boca Raton, USA: CRC Press.
- 11134 Balle, B., A. Quattoni, and X. Carreras (2011). A spectral learning algorithm for finite state
11135 transducers. In *Proceedings of the European Conference on Machine Learning and Principles
11136 and Practice of Knowledge Discovery in Databases (ECML)*, pp. 156–171.
- 11137 Banarescu, L., C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight,
11138 P. Koehn, M. Palmer, and N. Schneider (2013, August). Abstract meaning represen-
11139 tation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interopera-
11140 bility with Discourse*, Sofia, Bulgaria, pp. 178–186. Association for Computational
11141 Linguistics.
- 11142 Banko, M., M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni (2007). Open
11143 information extraction from the web. In *Proceedings of the International Joint Conference
11144 on Artificial Intelligence (IJCAI)*, pp. 2670–2676.
- 11145 Bansal, N., A. Blum, and S. Chawla (2004). Correlation clustering. *Machine Learning* 56(1–
11146 3), 89–113.
- 11147 Barber, D. (2012). *Bayesian reasoning and machine learning*. Cambridge University Press.
- 11148 Barman, U., A. Das, J. Wagner, and J. Foster (2014, October). Code mixing: A challenge for
11149 language identification in the language of social media. In *Proceedings of the First Work-
11150 shop on Computational Approaches to Code Switching*, Doha, Qatar, pp. 13–23. Association
11151 for Computational Linguistics.

- 11152 Baron, A. and P. Rayson (2008). Vard2: A tool for dealing with spelling variation in his-
11153 torical corpora. In *Postgraduate conference in corpus linguistics*.
- 11154 Baroni, M., R. Bernardi, and R. Zamparelli (2014). Frege in space: A program for compo-
11155 sitional distributional semantics. *Linguistic Issues in Language Technologies*.
- 11156 Barzilay, R. and M. Lapata (2008, mar). Modeling local coherence: An Entity-Based ap-
11157 proach. *Computational Linguistics* 34(1), 1–34.
- 11158 Barzilay, R. and K. R. McKeown (2005). Sentence fusion for multidocument news summa-
11159 rization. *Computational Linguistics* 31(3), 297–328.
- 11160 Beesley, K. R. and L. Karttunen (2003). *Finite-state morphology*. Stanford, CA: Center for
11161 the Study of Language and Information.
- 11162 Bejan, C. A. and S. Harabagiu (2014). Unsupervised event coreference resolution. *Compu-
11163 tational Linguistics* 40(2), 311–347.
- 11164 Bell, E. T. (1934). Exponential numbers. *The American Mathematical Monthly* 41(7), 411–419.
- 11165 Bender, E. M. (2013, jun). *Linguistic Fundamentals for Natural Language Processing: 100
11166 Essentials from Morphology and Syntax*, Volume 6 of *Synthesis Lectures on Human Language
11167 Technologies*. Morgan & Claypool Publishers.
- 11168 Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). Scheduled sampling for sequence
11169 prediction with recurrent neural networks. In *Neural Information Processing Systems
11170 (NIPS)*, pp. 1171–1179.
- 11171 Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin (2003). A neural probabilistic language
11172 model. *The Journal of Machine Learning Research* 3, 1137–1155.
- 11173 Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gra-
11174 dient descent is difficult. *IEEE Transactions on Neural Networks* 5(2), 157–166.
- 11175 Bengtson, E. and D. Roth (2008). Understanding the value of features for coreference
11176 resolution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11177 pp. 294–303.
- 11178 Benjamini, Y. and Y. Hochberg (1995). Controlling the false discovery rate: a practical and
11179 powerful approach to multiple testing. *Journal of the Royal Statistical Society. Series B
11180 (Methodological)*, 289–300.
- 11181 Berant, J., A. Chou, R. Frostig, and P. Liang (2013). Semantic parsing on freebase from
11182 question-answer pairs. In *Proceedings of Empirical Methods for Natural Language Process-
11183 ing (EMNLP)*, pp. 1533–1544.

- 11184 Berant, J., V. Srikumar, P.-C. Chen, A. Vander Linden, B. Harding, B. Huang, P. Clark, and
 11185 C. D. Manning (2014). Modeling biological processes for reading comprehension. In
 11186 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11187 Berg-Kirkpatrick, T., A. Bouchard-Côté, J. DeNero, and D. Klein (2010). Painless unsuper-
 11188 vised learning with features. In *Proceedings of the North American Chapter of the Associa-*
 11189 *tion for Computational Linguistics (NAACL)*, pp. 582–590.
- 11190 Berg-Kirkpatrick, T., D. Burkett, and D. Klein (2012). An empirical investigation of sta-
 11191 tistical significance in NLP. In *Proceedings of Empirical Methods for Natural Language*
 11192 *Processing (EMNLP)*, pp. 995–1005.
- 11193 Berger, A. L., V. J. D. Pietra, and S. A. D. Pietra (1996). A maximum entropy approach to
 11194 natural language processing. *Computational linguistics* 22(1), 39–71.
- 11195 Bergsma, S., D. Lin, and R. Goebel (2008). Distributional identification of non-referential
 11196 pronouns. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 10–18.
- 11197 Bernardi, R., R. Cakici, D. Elliott, A. Erdem, E. Erdem, N. Ikizler-Cinbis, F. Keller, A. Mus-
 11198 cat, and B. Plank (2016). Automatic description generation from images: A survey of
 11199 models, datasets, and evaluation measures. *Journal of Artificial Intelligence Research* 55,
 11200 409–442.
- 11201 Bertsekas, D. P. (2012). Incremental gradient, subgradient, and proximal methods for
 11202 convex optimization: A survey. See Sra et al. (2012).
- 11203 Bhatia, P., R. Guthrie, and J. Eisenstein (2016). Morphological priors for probabilistic neu-
 11204 ral word embeddings. In *Proceedings of Empirical Methods for Natural Language Processing*
 11205 (*EMNLP*).
- 11206 Bhatia, P., Y. Ji, and J. Eisenstein (2015). Better document-level sentiment analysis from
 11207 rst discourse parsing. In *Proceedings of Empirical Methods for Natural Language Processing*
 11208 (*EMNLP*).
- 11209 Biber, D. (1991). *Variation across speech and writing*. Cambridge University Press.
- 11210 Bird, S., E. Klein, and E. Loper (2009). *Natural language processing with Python*. California:
 11211 O'Reilly Media.
- 11212 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- 11213 Björkelund, A. and P. Nugues (2011). Exploring lexicalized features for coreference reso-
 11214 lution. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 45–50.
- 11215 Blackburn, P. and J. Bos (2005). *Representation and inference for natural language: A first*
 11216 *course in computational semantics*. CSLI.

- 11217 Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM* 55(4), 77–84.
- 11218 Blei, D. M. (2014). Build, compute, critique, repeat: Data analysis with latent variable
11219 models. *Annual Review of Statistics and Its Application* 1, 203–232.
- 11220 Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent dirichlet allocation. *the Journal of
11221 machine Learning research* 3, 993–1022.
- 11222 Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and
11223 blenders: Domain adaptation for sentiment classification. In *Proceedings of the Associa-
11224 tion for Computational Linguistics (ACL)*, pp. 440–447.
- 11225 Blum, A. and T. Mitchell (1998). Combining labeled and unlabeled data with co-training.
11226 In *Proceedings of the Conference On Learning Theory (COLT)*, pp. 92–100.
- 11227 Bobrow, D. G., R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd
11228 (1977). Gus, a frame-driven dialog system. *Artificial intelligence* 8(2), 155–173.
- 11229 Bochnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction.
11230 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
11231 89–97.
- 11232 Boitet, C. (1988). Pros and cons of the pivot and transfer approaches in multilingual ma-
11233 chine translation. *Readings in machine translation*, 273–279.
- 11234 Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov (2017). Enriching word vectors with
11235 subword information. *Transactions of the Association for Computational Linguistics* 5, 135–
11236 146.
- 11237 Bollacker, K., C. Evans, P. Paritosh, T. Sturge, and J. Taylor (2008). Freebase: a collabora-
11238 tively created graph database for structuring human knowledge. In *Proceedings of the
11239 ACM International Conference on Management of Data (SIGMOD)*, pp. 1247–1250. AcM.
- 11240 Bolukbasi, T., K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai (2016). Man is to
11241 computer programmer as woman is to homemaker? debiasing word embeddings. In *Neural
11242 Information Processing Systems (NIPS)*, pp. 4349–4357.
- 11243 Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating
11244 embeddings for modeling multi-relational data. In *Neural Information Processing Systems
(NIPS)*, pp. 2787–2795.
- 11246 Bordes, A., J. Weston, R. Collobert, Y. Bengio, et al. (2011). Learning structured embed-
11247 dings of knowledge bases. In *Proceedings of the National Conference on Artificial Intelligence
(AAAI)*, pp. 301–306.

- 11249 Borges, J. L. (1993). *Other Inquisitions 1937–1952*. University of Texas Press. Translated by
 11250 Ruth L. C. Simms.
- 11251 Botha, J. A. and P. Blunsom (2014). Compositional morphology for word representations
 11252 and language modelling. In *Proceedings of the International Conference on Machine Learn-*
 11253 *ing (ICML)*.
- 11254 Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*,
 11255 pp. 421–436. Springer.
- 11256 Bottou, L., F. E. Curtis, and J. Nocedal (2016). Optimization methods for large-scale ma-
 11257 chine learning. *arXiv preprint arXiv:1606.04838*.
- 11258 Bowman, S. R., L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio (2016). Gen-
 11259 erating sentences from a continuous space. In *Proceedings of the Conference on Natural*
 11260 *Language Learning (CoNLL)*, pp. 10–21.
- 11261 boyd, d. and K. Crawford (2012). Critical questions for big data. *Information, Communica-*
 11262 *tion & Society* 15(5), 662–679.
- 11263 Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. New York: Cambridge Uni-
 11264 versity Press.
- 11265 Boydston, A. E. (2013). *Making the news: Politics, the media, and agenda setting*. University
 11266 of Chicago Press.
- 11267 Branavan, S., H. Chen, J. Eisenstein, and R. Barzilay (2009). Learning document-level
 11268 semantic properties from free-text annotations. *Journal of Artificial Intelligence Re-*
 11269 *search* 34(2), 569–603.
- 11270 Branavan, S. R., H. Chen, L. S. Zettlemoyer, and R. Barzilay (2009). Reinforcement learning
 11271 for mapping instructions to actions. In *Proceedings of the Association for Computational*
 11272 *Linguistics (ACL)*, pp. 82–90.
- 11273 Brants, T. and A. Franz (2006). The Google 1T 5-gram corpus. LDC2006T13.
- 11274 Braud, C., O. Lacroix, and A. Søgaard (2017). Does syntax help discourse segmenta-
 11275 tion? not so much. In *Proceedings of Empirical Methods for Natural Language Processing*
 11276 (*EMNLP*), pp. 2432–2442.
- 11277 Brennan, S. E., M. W. Friedman, and C. J. Pollard (1987). A centering approach to pro-
 11278 nouns. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 155–162.
- 11279 Briscoe, T. (2011). Introduction to formal semantics for natural language.
 11280 <https://www.cl.cam.ac.uk/teaching/1011/L107/semantics.pdf>.

- 11281 Brown, P. F., J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer,
11282 and P. S. Roossin (1990). A statistical approach to machine translation. *Computational
11283 linguistics* 16(2), 79–85.
- 11284 Brown, P. F., P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai (1992). Class-based
11285 n-gram models of natural language. *Computational linguistics* 18(4), 467–479.
- 11286 Brown, P. F., V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer (1993). The mathematics
11287 of statistical machine translation: Parameter estimation. *Computational linguistics* 19(2),
11288 263–311.
- 11289 Brun, C. and C. Roux (2014). Décomposition des “hash tags” pour l’amélioration de la
11290 classification en polarité des “tweets”. *Proceedings of Traitement Automatique des Langues
11291 Naturelles*, 473–478.
- 11292 Bruni, E., N.-K. Tran, and M. Baroni (2014). Multimodal distributional semantics. *Journal
11293 of Artificial Intelligence Research* 49(2014), 1–47.
- 11294 Bullinaria, J. A. and J. P. Levy (2007). Extracting semantic representations from word co-
11295 occurrence statistics: A computational study. *Behavior research methods* 39(3), 510–526.
- 11296 Bunescu, R. C. and R. J. Mooney (2005). A shortest path dependency kernel for relation
11297 extraction. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11298 pp. 724–731.
- 11299 Bunescu, R. C. and M. Pasca (2006). Using encyclopedic knowledge for named entity
11300 disambiguation. In *Proceedings of the European Chapter of the Association for Computational
11301 Linguistics (EACL)*, pp. 9–16.
- 11302 Burstein, J., D. Marcu, and K. Knight (2003). Finding the WRITE stuff: Automatic identi-
11303 fication of discourse structure in student essays. *IEEE Intelligent Systems* 18(1), 32–39.
- 11304 Burstein, J., J. Tetreault, and S. Andreyev (2010). Using entity-based features to model
11305 coherence in student essays. In *Human language technologies: The 2010 annual conference
11306 of the North American chapter of the Association for Computational Linguistics*, pp. 681–684.
11307 Association for Computational Linguistics.
- 11308 Burstein, J., J. Tetreault, and M. Chodorow (2013). Holistic discourse coherence annotation
11309 for noisy essay writing. *Dialogue & Discourse* 4(2), 34–52.
- 11310 Cai, Q. and A. Yates (2013). Large-scale semantic parsing via schema matching and lexicon
11311 extension. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 423–
11312 433.

- 11313 Caliskan, A., J. J. Bryson, and A. Narayanan (2017). Semantics derived automatically from
 11314 language corpora contain human-like biases. *Science* 356(6334), 183–186.
- 11315 Canny, J. (1987). A computational approach to edge detection. In *Readings in Computer*
 11316 *Vision*, pp. 184–203. Elsevier.
- 11317 Cappé, O. and E. Moulines (2009). On-line expectation–maximization algorithm for latent
 11318 data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71(3),
 11319 593–613.
- 11320 Carbonell, J. and J. Goldstein (1998). The use of mmr, diversity-based reranking for re-
 11321 ordering documents and producing summaries. In *Proceedings of ACM SIGIR conference*
 11322 *on Research and development in information retrieval*, pp. 335–336.
- 11323 Carbonell, J. R. (1970). Mixed-initiative man-computer instructional dialogues. Technical
 11324 report, BOLT BERANEK AND NEWMAN INC CAMBRIDGE MASS.
- 11325 Cardie, C. and K. Wagstaff (1999). Noun phrase coreference as clustering. In *Proceedings*
 11326 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 82–89.
- 11327 Carletta, J. (1996). Assessing agreement on classification tasks: the kappa statistic. *Com-*
 11328 *putational linguistics* 22(2), 249–254.
- 11329 Carletta, J. (2007). Unleashing the killer corpus: experiences in creating the multi-
 11330 everything ami meeting corpus. *Language Resources and Evaluation* 41(2), 181–190.
- 11331 Carlson, L. and D. Marcu (2001). Discourse tagging reference manual. Technical Report
 11332 ISI-TR-545, Information Sciences Institute.
- 11333 Carlson, L., M. E. Okurowski, and D. Marcu (2002). RST discourse treebank. Linguistic
 11334 Data Consortium, University of Pennsylvania.
- 11335 Carpenter, B. (1997). *Type-logical semantics*. Cambridge, MA: MIT Press.
- 11336 Carreras, X., M. Collins, and T. Koo (2008). Tag, dynamic programming, and the percep-
 11337 tron for efficient, feature-rich parsing. In *Proceedings of the Conference on Natural Language*
 11338 *Learning (CoNLL)*, pp. 9–16.
- 11339 Carreras, X. and L. Màrquez (2005). Introduction to the conll-2005 shared task: Semantic
 11340 role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language*
 11341 *Learning*, pp. 152–164. Association for Computational Linguistics.
- 11342 Carroll, L. (1865). *Alice's Adventures in Wonderland*. London: Macmillan.
- 11343 Carroll, L. (1917). *Through the looking glass: And what Alice found there*. Chicago: Rand,
 11344 McNally.

- 11345 Chambers, N. and D. Jurafsky (2008). Jointly combining implicit constraints improves
11346 temporal ordering. In *Proceedings of Empirical Methods for Natural Language Processing*
11347 (*EMNLP*), pp. 698–706.
- 11348 Chang, K.-W., A. Krishnamurthy, A. Agarwal, H. Daume III, and J. Langford (2015).
11349 Learning to search better than your teacher. In *Proceedings of the International Conference*
11350 *on Machine Learning (ICML)*.
- 11351 Chang, M.-W., L. Ratinov, and D. Roth (2007). Guiding semi-supervision with constraint-
11352 driven learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11353 280–287.
- 11354 Chang, M.-W., L.-A. Ratinov, N. Rizzolo, and D. Roth (2008). Learning and inference with
11355 constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
11356 1513–1518.
- 11357 Chapman, W. W., W. Bridewell, P. Hanbury, G. F. Cooper, and B. G. Buchanan (2001). A
11358 simple algorithm for identifying negated findings and diseases in discharge summaries.
11359 *Journal of biomedical informatics* 34(5), 301–310.
- 11360 Charniak, E. (1997). Statistical techniques for natural language parsing. *AI magazine* 18(4),
11361 33–43.
- 11362 Charniak, E. and M. Johnson (2005). Coarse-to-fine n-best parsing and maxent discriminative
11363 reranking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11364 173–180.
- 11365 Chelba, C. and A. Acero (2006). Adaptation of maximum entropy capitalizer: Little data
11366 can help a lot. *Computer Speech & Language* 20(4), 382–399.
- 11367 Chelba, C., T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson (2013).
11368 One billion word benchmark for measuring progress in statistical language modeling.
11369 *arXiv preprint arXiv:1312.3005*.
- 11370 Chen, D., J. Bolton, and C. D. Manning (2016). A thorough examination of the CNN/Daily
11371 Mail reading comprehension task. In *Proceedings of the Association for Computational*
11372 *Linguistics (ACL)*.
- 11373 Chen, D. and C. D. Manning (2014). A fast and accurate dependency parser using neural
11374 networks. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
11375 pp. 740–750.
- 11376 Chen, D. L. and R. J. Mooney (2008). Learning to sportscast: a test of grounded language
11377 acquisition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.
11378 128–135.

- 11379 Chen, H., S. Branavan, R. Barzilay, and D. R. Karger (2009). Content modeling using latent
 11380 permutations. *Journal of Artificial Intelligence Research* 36(1), 129–163.
- 11381 Chen, M., Z. Xu, K. Weinberger, and F. Sha (2012). Marginalized denoising autoencoders
 11382 for domain adaptation. In *Proceedings of the International Conference on Machine Learning*
 11383 (*ICML*).
- 11384 Chen, M. X., O. Firat, A. Bapna, M. Johnson, W. Macherey, G. Foster, L. Jones, N. Parmar,
 11385 M. Schuster, Z. Chen, Y. Wu, and M. Hughes (2018). The best of both worlds: Combin-
 11386 ing recent advances in neural machine translation. In *Proceedings of the Association for*
 11387 *Computational Linguistics (ACL)*.
- 11388 Chen, S. F. and J. Goodman (1999). An empirical study of smoothing techniques for lan-
 11389 guage modeling. *Computer Speech & Language* 13(4), 359–393.
- 11390 Chen, T. and C. Guestrin (2016). Xgboost: A scalable tree boosting system. In *Proceedings*
 11391 *of Knowledge Discovery and Data Mining (KDD)*, pp. 785–794.
- 11392 Chen, X., X. Qiu, C. Zhu, P. Liu, and X. Huang (2015). Long short-term memory neural
 11393 networks for chinese word segmentation. In *Proceedings of Empirical Methods for Natural*
 11394 *Language Processing (EMNLP)*, pp. 1197–1206.
- 11395 Chen, Y., S. Gilroy, A. Malletti, K. Knight, and J. May (2018). Recurrent neural networks
 11396 as weighted language recognizers. In *Proceedings of the North American Chapter of the*
 11397 *Association for Computational Linguistics (NAACL)*.
- 11398 Chen, Z. and H. Ji (2009). Graph-based event coreference resolution. In *Proceedings of*
 11399 *the 2009 Workshop on Graph-based Methods for Natural Language Processing*, pp. 54–57.
 11400 Association for Computational Linguistics.
- 11401 Cheng, X. and D. Roth (2013). Relational inference for wikification. In *Proceedings of*
 11402 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1787–1796.
- 11403 Chiang, D. (2007). Hierarchical phrase-based translation. *Computational Linguistics* 33(2),
 11404 201–228.
- 11405 Chiang, D., J. Graehl, K. Knight, A. Pauls, and S. Ravi (2010). Bayesian inference for
 11406 finite-state transducers. In *Proceedings of the North American Chapter of the Association for*
 11407 *Computational Linguistics (NAACL)*, pp. 447–455.
- 11408 Chinchor, N. and P. Robinson (1997). Muc-7 named entity task definition. In *Proceedings*
 11409 *of the 7th Conference on Message Understanding*, Volume 29.
- 11410 Cho, K. (2015). Natural language understanding with distributed representation.
 11411 *CoRR abs/1511.07916*.

- 11412 Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and
11413 Y. Bengio (2014). Learning phrase representations using rnn encoder-decoder for sta-
11414 tistical machine translation. In *Proceedings of Empirical Methods for Natural Language*
11415 *Processing (EMNLP)*.
- 11416 Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton & Co.
- 11417 Chomsky, N. (1982). *Some concepts and consequences of the theory of government and binding*,
11418 Volume 6. MIT press.
- 11419 Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss
11420 surfaces of multilayer networks. In *Proceedings of Artificial Intelligence and Statistics (AIS-
11421 TATS)*, pp. 192–204.
- 11422 Christodoulopoulos, C., S. Goldwater, and M. Steedman (2010). Two decades of unsuper-
11423 vised pos induction: How far have we come? In *Proceedings of Empirical Methods for
11424 Natural Language Processing (EMNLP)*, pp. 575–584.
- 11425 Chu, Y.-J. and T.-H. Liu (1965). On shortest arborescence of a directed graph. *Scientia
11426 Sinica* 14(10), 1396–1400.
- 11427 Chung, C. and J. W. Pennebaker (2007). The psychological functions of function words.
11428 In K. Fiedler (Ed.), *Social communication*, pp. 343–359. New York and Hove: Psychology
11429 Press.
- 11430 Church, K. (2011). A pendulum swung too far. *Linguistic Issues in Language Technology* 6(5),
11431 1–27.
- 11432 Church, K. W. (2000). Empirical estimates of adaptation: the chance of two Noriega-
11433 s is closer to $p/2$ than p^2 . In *Proceedings of the International Conference on Computational
11434 Linguistics (COLING)*, pp. 180–186.
- 11435 Church, K. W. and P. Hanks (1990). Word association norms, mutual information, and
11436 lexicography. *Computational linguistics* 16(1), 22–29.
- 11437 Ciaramita, M. and M. Johnson (2003). Supersense tagging of unknown nouns in wordnet.
11438 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 168–
11439 175.
- 11440 Clark, K. and C. D. Manning (2015). Entity-centric coreference resolution with model
11441 stacking. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1405–
11442 1415.
- 11443 Clark, K. and C. D. Manning (2016). Improving coreference resolution by learning entity-
11444 level distributed representations. In *Proceedings of the Association for Computational Lin-
11445 guistics (ACL)*.

- 11446 Clark, P. (2015). Elementary school science and math tests as a driver for ai: take the aristo
 11447 challenge! In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp.
 11448 4019–4021.
- 11449 Clarke, J., D. Goldwasser, M.-W. Chang, and D. Roth (2010). Driving semantic parsing
 11450 from the world’s response. In *Proceedings of the Conference on Natural Language Learning*
 11451 (*CoNLL*), pp. 18–27.
- 11452 Clarke, J. and M. Lapata (2008). Global inference for sentence compression: An integer
 11453 linear programming approach. *Journal of Artificial Intelligence Research* 31, 399–429.
- 11454 Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychologi-*
 11455 *cal measurement* 20(1), 37–46.
- 11456 Cohen, S. (2016). *Bayesian analysis in natural language processing*. Synthesis Lectures on
 11457 Human Language Technologies. San Rafael, CA: Morgan & Claypool Publishers.
- 11458 Cohen, S. B., K. Stratos, M. Collins, D. P. Foster, and L. Ungar (2014). Spectral learning of
 11459 latent-variable PCFGs: Algorithms and sample complexity. *Journal of Machine Learning*
 11460 *Research* 15, 2399–2449.
- 11461 Collier, N., C. Nobata, and J.-i. Tsujii (2000). Extracting the names of genes and gene
 11462 products with a hidden markov model. In *Proceedings of the International Conference on*
 11463 *Computational Linguistics (COLING)*, pp. 201–207.
- 11464 Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceed-*
 11465 *ings of the Association for Computational Linguistics (ACL)*, pp. 16–23.
- 11466 Collins, M. (2002). Discriminative training methods for hidden markov models: theory
 11467 and experiments with perceptron algorithms. In *Proceedings of Empirical Methods for*
 11468 *Natural Language Processing (EMNLP)*, pp. 1–8.
- 11469 Collins, M. (2013). Notes on natural language processing. <http://www.cs.columbia.edu/~mcollins/notes-spring2013.html>.
- 11471 Collins, M. and T. Koo (2005). Discriminative reranking for natural language parsing.
 11472 *Computational Linguistics* 31(1), 25–70.
- 11473 Collins, M. and B. Roark (2004). Incremental parsing with the perceptron algorithm. In
 11474 *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pp.
 11475 111. Association for Computational Linguistics.
- 11476 Collobert, R., K. Kavukcuoglu, and C. Farabet (2011). Torch7: A matlab-like environment
 11477 for machine learning. Technical Report EPFL-CONF-192376, EPFL.

- 11478 Collobert, R. and J. Weston (2008). A unified architecture for natural language processing:
11479 Deep neural networks with multitask learning. In *Proceedings of the International
11480 Conference on Machine Learning (ICML)*, pp. 160–167.
- 11481 Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). Natural
11482 language processing (almost) from scratch. *Journal of Machine Learning Research* 12,
11483 2493–2537.
- 11484 Colton, S., J. Goodwin, and T. Veale (2012). Full-face poetry generation. In *Proceedings of
11485 the International Conference on Computational Creativity*, pp. 95–102.
- 11486 Conneau, A., D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). Supervised learning
11487 of universal sentence representations from natural language inference data. In *Proceed-
11488 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 681–691.
- 11489 Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to algorithms*
11490 (third ed.). MIT press.
- 11491 Cotterell, R., H. Schütze, and J. Eisner (2016). Morphological smoothing and extrapolation
11492 of word embeddings. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11493 pp. 1651–1660.
- 11494 Covello, L., Y. Sohn, A. D. Kramer, C. Marlow, M. Franceschetti, N. A. Christakis, and
11495 J. H. Fowler (2014). Detecting emotional contagion in massive social networks. *PloS
11496 one* 9(3), e90315.
- 11497 Covington, M. A. (2001). A fundamental algorithm for dependency parsing. In *Proceedings
11498 of the 39th annual ACM southeast conference*, pp. 95–102.
- 11499 Crammer, K., O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer (2006, December).
11500 Online passive-aggressive algorithms. *The Journal of Machine Learning Research* 7, 551–
11501 585.
- 11502 Crammer, K. and Y. Singer (2001). Pranking with ranking. In *Neural Information Processing
11503 Systems (NIPS)*, pp. 641–647.
- 11504 Crammer, K. and Y. Singer (2003). Ultraconservative online algorithms for multiclass
11505 problems. *The Journal of Machine Learning Research* 3, 951–991.
- 11506 Creutz, M. and K. Lagus (2007). Unsupervised models for morpheme segmentation and
11507 morphology learning. *ACM Transactions on Speech and Language Processing (TSLP)* 4(1),
11508 3.
- 11509 Cross, J. and L. Huang (2016). Span-based constituency parsing with a structure-label
11510 system and provably optimal dynamic oracles. In *Proceedings of Empirical Methods for
11511 Natural Language Processing (EMNLP)*, pp. 1–11.

- 11512 Cucerzan, S. (2007). Large-scale named entity disambiguation based on wikipedia data.
 11513 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11514 Cui, H., R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua (2005). Question answering passage
 11515 retrieval using dependency relations. In *Proceedings of the 28th annual international ACM*
 11516 *SIGIR conference on Research and development in information retrieval*, pp. 400–407. ACM.
- 11517 Cui, Y., Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu (2017). Attention-over-attention neural
 11518 networks for reading comprehension. In *Proceedings of the Association for Computational*
 11519 *Linguistics (ACL)*.
- 11520 Culotta, A. and J. Sorensen (2004). Dependency tree kernels for relation extraction. In
 11521 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11522 Culotta, A., M. Wick, and A. McCallum (2007). First-order probabilistic models for coref-
 11523 erence resolution. In *Proceedings of the North American Chapter of the Association for Com-*
 11524 *putational Linguistics (NAACL)*, pp. 81–88.
- 11525 Curry, H. B. and R. Feys (1958). *Combinatory Logic*, Volume I. Amsterdam: North Holland.
- 11526 Danescu-Niculescu-Mizil, C., M. Sudhof, D. Jurafsky, J. Leskovec, and C. Potts (2013). A
 11527 computational approach to politeness with application to social factors. In *Proceedings*
 11528 *of the Association for Computational Linguistics (ACL)*, pp. 250–259.
- 11529 Das, D., D. Chen, A. F. Martins, N. Schneider, and N. A. Smith (2014). Frame-semantic
 11530 parsing. *Computational Linguistics* 40(1), 9–56.
- 11531 Daumé III, H. (2007). Frustratingly easy domain adaptation. In *Proceedings of the Associa-*
 11532 *tion for Computational Linguistics (ACL)*.
- 11533 Daumé III, H., J. Langford, and D. Marcu (2009). Search-based structured prediction.
 11534 *Machine learning* 75(3), 297–325.
- 11535 Daumé III, H. and D. Marcu (2005). A large-scale exploration of effective global features
 11536 for a joint entity detection and tracking model. In *Proceedings of Empirical Methods for*
 11537 *Natural Language Processing (EMNLP)*, pp. 97–104.
- 11538 Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Iden-
 11539 tifying and attacking the saddle point problem in high-dimensional non-convex opti-
 11540 mization. In *Neural Information Processing Systems (NIPS)*, pp. 2933–2941.
- 11541 Davidson, D. (1967). The logical form of action sentences. In N. Rescher (Ed.), *The Logic of*
 11542 *Decision and Action*. Pittsburgh: University of Pittsburgh Press.

- 11543 De Gispert, A. and J. B. Marino (2006). Catalan-english statistical machine translation
11544 without parallel corpus: bridging through spanish. In *Proc. of 5th International Conference*
11545 on *Language Resources and Evaluation (LREC)*, pp. 65–68. Citeseer.
- 11546 De Marneffe, M.-C. and C. D. Manning (2008). The stanford typed dependencies represen-
11547 tation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain*
11548 *Parser Evaluation*, pp. 1–8. Association for Computational Linguistics.
- 11549 Dean, J. and S. Ghemawat (2008). Mapreduce: simplified data processing on large clusters.
11550 *Communications of the ACM* 51(1), 107–113.
- 11551 Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman (1990).
11552 Indexing by latent semantic analysis. *JASIS* 41(6), 391–407.
- 11553 Dehdari, J. (2014). *A Neurophysiologically-Inspired Statistical Language Model*. Ph. D. thesis,
11554 The Ohio State University.
- 11555 Deisenroth, M. P., A. A. Faisal, and C. S. Ong (2018). *Mathematics For Machine Learning*.
11556 Cambridge UP.
- 11557 Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). Maximum likelihood from incom-
11558 plete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodo-*
11559 *logical)*, 1–38.
- 11560 Denis, P. and J. Baldridge (2007). A ranking approach to pronoun resolution. In *IJCAI*.
- 11561 Denis, P. and J. Baldridge (2008). Specialized models and ranking for coreference resolu-
11562 tion. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*,
11563 EMNLP '08, Stroudsburg, PA, USA, pp. 660–669. Association for Computational Lin-
11564 guistics.
- 11565 Denis, P. and J. Baldridge (2009). Global joint models for coreference resolution and named
11566 entity classification. *Procesamiento del Lenguaje Natural* 42.
- 11567 Derrida, J. (1985). Des tours de babel. In J. Graham (Ed.), *Difference in translation*. Ithaca,
11568 NY: Cornell University Press.
- 11569 Dhingra, B., H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov (2017). Gated-attention
11570 readers for text comprehension. In *Proceedings of the Association for Computational Lin-*
11571 *guistics (ACL)*.
- 11572 Diaconis, P. and B. Skyrms (2017). *Ten Great Ideas About Chance*. Princeton University
11573 Press.
- 11574 Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classifica-
11575 tion learning algorithms. *Neural computation* 10(7), 1895–1923.

- 11576 Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple instance
 11577 problem with axis-parallel rectangles. *Artificial intelligence* 89(1), 31–71.
- 11578 Dimitrova, L., N. Ide, V. Petkevic, T. Erjavec, H. J. Kaalep, and D. Tufis (1998). Multext-
 11579 east: Parallel and comparable corpora and lexicons for six central and eastern european
 11580 languages. In *Proceedings of the 17th international conference on Computational linguistics-*
 11581 *Volume 1*, pp. 315–319. Association for Computational Linguistics.
- 11582 Doddington, G. R., A. Mitchell, M. A. Przybocki, L. A. Ramshaw, S. Strassel, and R. M.
 11583 Weischedel (2004). The automatic content extraction (ace) program-tasks, data, and
 11584 evaluation. In *Proceedings of the Language Resources and Evaluation Conference*, pp. 837–
 11585 840.
- 11586 dos Santos, C., B. Xiang, and B. Zhou (2015). Classifying relations by ranking with con-
 11587 volutional neural networks. In *Proceedings of the Association for Computational Linguistics*
 11588 (*ACL*), pp. 626–634.
- 11589 Dowty, D. (1991). Thematic proto-roles and argument selection. *Language*, 547–619.
- 11590 Dredze, M., P. McNamee, D. Rao, A. Gerber, and T. Finin (2010). Entity disambiguation
 11591 for knowledge base population. In *Proceedings of the 23rd International Conference on*
 11592 *Computational Linguistics*, pp. 277–285. Association for Computational Linguistics.
- 11593 Dredze, M., M. J. Paul, S. Bergsma, and H. Tran (2013). Carmen: A Twitter geolocation
 11594 system with applications to public health. In *AAAI workshop on expanding the boundaries*
 11595 *of health informatics using AI (HIAI)*, pp. 20–24.
- 11596 Dreyfus, H. L. (1992). *What computers still can't do: a critique of artificial reason*. MIT press.
- 11597 Du, L., W. Buntine, and M. Johnson (2013). Topic segmentation with a structured topic
 11598 model. In *Proceedings of the North American Chapter of the Association for Computational*
 11599 *Linguistics (NAACL)*, pp. 190–200.
- 11600 Duchi, J., E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learn-
 11601 ing and stochastic optimization. *The Journal of Machine Learning Research* 12, 2121–2159.
- 11602 Dunietz, J., L. Levin, and J. Carbonell (2017). The because corpus 2.0: Annotating causality
 11603 and overlapping relations. In *Proceedings of the Linguistic Annotation Workshop*.
- 11604 Durrett, G., T. Berg-Kirkpatrick, and D. Klein (2016). Learning-based single-document
 11605 summarization with compression and anaphoricity constraints. In *Proceedings of the*
 11606 *Association for Computational Linguistics (ACL)*, pp. 1998–2008.
- 11607 Durrett, G. and D. Klein (2013). Easy victories and uphill battles in coreference resolution.
 11608 In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

- 11609 Durrett, G. and D. Klein (2015). Neural crf parsing. In *Proceedings of the Association for
11610 Computational Linguistics (ACL)*.
- 11611 Dyer, C., M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith (2015). Transition-based
11612 dependency parsing with stack long short-term memory. In *Proceedings of the Association
11613 for Computational Linguistics (ACL)*, pp. 334–343.
- 11614 Dyer, C., A. Kuncoro, M. Ballesteros, and N. A. Smith (2016). Recurrent neural network
11615 grammars. In *Proceedings of the North American Chapter of the Association for Computational
11616 Linguistics (NAACL)*, pp. 199–209.
- 11617 Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of
11618 Standards B* 71(4), 233–240.
- 11619 Efron, B. and R. J. Tibshirani (1993). An introduction to the bootstrap: Monographs on
11620 statistics and applied probability, vol. 57. *New York and London: Chapman and Hall/CRC*.
- 11621 Eisenstein, J. (2009). Hierarchical text segmentation from multi-scale lexical cohesion. In
11622 *Proceedings of the North American Chapter of the Association for Computational Linguistics
11623 (NAACL)*.
- 11624 Eisenstein, J. and R. Barzilay (2008). Bayesian unsupervised topic segmentation. In *Pro-
11625 ceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 11626 Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial dis-
11627 cussion.
- 11628 Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. In *Advances
11629 in probabilistic and other parsing technologies*, pp. 29–61. Springer.
- 11630 Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceed-
11631 ings of the Association for Computational Linguistics (ACL)*, pp. 1–8.
- 11632 Eisner, J. (2016). Inside-outside and forward-backward algorithms are just backprop. In
11633 *Proceedings of the Workshop on Structured Prediction for NLP*, pp. 1–17.
- 11634 Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An explo-
11635 ration. In *Proceedings of the International Conference on Computational Linguistics (COL-
11636 ING)*, pp. 340–345.
- 11637 Ekman, P. (1992). Are there basic emotions? *Psychological Review* 99(3), 550–553.
- 11638 Elman, J. L. (1990). Finding structure in time. *Cognitive science* 14(2), 179–211.

- 11639 Elman, J. L., E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett
 11640 (1998). *Rethinking innateness: A connectionist perspective on development*, Volume 10. MIT
 11641 press.
- 11642 Elsner, M. and E. Charniak (2010). Disentangling chat. *Computational Linguistics* 36(3),
 11643 389–409.
- 11644 Esuli, A. and F. Sebastiani (2006). Sentiwordnet: A publicly available lexical resource for
 11645 opinion mining. In *LREC*, Volume 6, pp. 417–422. Citeseer.
- 11646 Etzioni, O., A. Fader, J. Christensen, S. Soderland, and M. Mausam (2011). Open informa-
 11647 tion extraction: The second generation. In *Proceedings of the International Joint Conference*
 11648 on *Artificial Intelligence (IJCAI)*, pp. 3–10.
- 11649 Faruqui, M., J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith (2015). Retrofitting
 11650 word vectors to semantic lexicons. In *Proceedings of the North American Chapter of the*
 11651 *Association for Computational Linguistics (NAACL)*.
- 11652 Faruqui, M. and C. Dyer (2014). Improving vector space word representations using mul-
 11653 tilingual correlation. In *Proceedings of the European Chapter of the Association for Compu-*
 11654 *tational Linguistics (EACL)*, pp. 462–471.
- 11655 Faruqui, M., R. McDonald, and R. Soricut (2016). Morpho-syntactic lexicon generation
 11656 using graph-based semi-supervised learning. *Transactions of the Association for Compu-*
 11657 *tational Linguistics* 4, 1–16.
- 11658 Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer (2016, August). Problems with evaluation
 11659 of word embeddings using word similarity tasks. In *Proceedings of the 1st Workshop on*
 11660 *Evaluating Vector-Space Representations for NLP*, Berlin, Germany, pp. 30–35. Association
 11661 for Computational Linguistics.
- 11662 Fellbaum, C. (2010). *WordNet*. Springer.
- 11663 Feng, V. W., Z. Lin, and G. Hirst (2014). The impact of deep hierarchical discourse struc-
 11664 tures in the evaluation of text coherence. In *Proceedings of the International Conference on*
 11665 *Computational Linguistics (COLING)*, pp. 940–949.
- 11666 Feng, X., L. Huang, D. Tang, H. Ji, B. Qin, and T. Liu (2016). A language-independent
 11667 neural network for event detection. In *Proceedings of the Association for Computational*
 11668 *Linguistics (ACL)*, pp. 66–71.
- 11669 Fernandes, E. R., C. N. dos Santos, and R. L. Milidiú (2014). Latent trees for coreference
 11670 resolution. *Computational Linguistics*.

- 11671 Ferrucci, D., E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W.
11672 Murdock, E. Nyberg, J. Prager, et al. (2010). Building Watson: An overview of the
11673 DeepQA project. *AI magazine* 31(3), 59–79.
- 11674 Ficler, J. and Y. Goldberg (2017, September). Controlling linguistic style aspects in neural
11675 language generation. In *Proceedings of the Workshop on Stylistic Variation*, Copenhagen,
11676 Denmark, pp. 94–104. Association for Computational Linguistics.
- 11677 Filippova, K. and M. Strube (2008). Sentence fusion via dependency graph compression.
11678 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 177–
11679 185.
- 11680 Fillmore, C. J. (1968). The case for case. In E. Bach and R. Harms (Eds.), *Universals in
11681 linguistic theory*. Holt, Rinehart, and Winston.
- 11682 Fillmore, C. J. (1976). Frame semantics and the nature of language. *Annals of the New York
11683 Academy of Sciences* 280(1), 20–32.
- 11684 Fillmore, C. J. and C. Baker (2009). A frames approach to semantic analysis. In *The Oxford
11685 Handbook of Linguistic Analysis*. Oxford University Press.
- 11686 Finkel, J. R., T. Grenager, and C. Manning (2005). Incorporating non-local information
11687 into information extraction systems by gibbs sampling. In *Proceedings of the Association
11688 for Computational Linguistics (ACL)*, pp. 363–370.
- 11689 Finkel, J. R., T. Grenager, and C. D. Manning (2007). The infinite tree. In *Proceedings of the
11690 Association for Computational Linguistics (ACL)*, pp. 272–279.
- 11691 Finkel, J. R., A. Kleeman, and C. D. Manning (2008). Efficient, feature-based, conditional
11692 random field parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11693 pp. 959–967.
- 11694 Finkel, J. R. and C. Manning (2009). Hierarchical bayesian domain adaptation. In *Proceed-
11695 ings of the North American Chapter of the Association for Computational Linguistics (NAACL)*,
11696 pp. 602–610.
- 11697 Finkel, J. R. and C. D. Manning (2008). Enforcing transitivity in coreference resolution.
11698 In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics
11699 on Human Language Technologies: Short Papers*, pp. 45–48. Association for Computational
11700 Linguistics.
- 11701 Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin
11702 (2002). Placing search in context: The concept revisited. *ACM Transactions on Information
11703 Systems* 20(1), 116–131.

- 11704 Firth, J. R. (1957). *Papers in Linguistics 1934-1951*. Oxford University Press.
- 11705 Flanigan, J., S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith (2014). A discriminative graph-based parser for the abstract meaning representation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1426–1436.
- 11708 Foltz, P. W., W. Kintsch, and T. K. Landauer (1998). The measurement of textual coherence with latent semantic analysis. *Discourse processes* 25(2-3), 285–307.
- 11710 Fordyce, C. S. (2007). Overview of the iwslt 2007 evaluation campaign. In *International Workshop on Spoken Language Translation (IWSLT) 2007*.
- 11712 Forsyth, E. N. and C. H. Martell (2007). Lexical and discourse analysis of online chat dialog. In *International Conference on Semantic Computing*, pp. 19–26. IEEE.
- 11714 Fort, K., G. Adda, and K. B. Cohen (2011). Amazon mechanical turk: Gold mine or coal mine? *Computational Linguistics* 37(2), 413–420.
- 11716 Fox, H. (2002). Phrasal cohesion and statistical machine translation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 304–3111.
- 11718 Francis, W. and H. Kucera (1982). *Frequency analysis of English usage*. Houghton Mifflin Company.
- 11720 Francis, W. N. (1964). A standard sample of present-day English for use with digital computers. Report to the U.S Office of Education on Cooperative Research Project No. E-007.
- 11723 Freund, Y. and R. E. Schapire (1999). Large margin classification using the perceptron algorithm. *Machine learning* 37(3), 277–296.
- 11725 Fromkin, V., R. Rodman, and N. Hyams (2013). *An introduction to language*. Cengage Learning.
- 11727 Fundel, K., R. Küffner, and R. Zimmer (2007). Relex – relation extraction using dependency parse trees. *Bioinformatics* 23(3), 365–371.
- 11729 Gabow, H. N., Z. Galil, T. Spencer, and R. E. Tarjan (1986). Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica* 6(2), 109–122.
- 11732 Gabrilovich, E. and S. Markovitch (2007). Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Volume 7, pp. 1606–1611.
- 11735 Gage, P. (1994). A new algorithm for data compression. *The C Users Journal* 12(2), 23–38.

- 11736 Gale, W. A., K. W. Church, and D. Yarowsky (1992). One sense per discourse. In *Proceedings of the workshop on Speech and Natural Language*, pp. 233–237. Association for
11737 Computational Linguistics.
- 11739 Galley, M., M. Hopkins, K. Knight, and D. Marcu (2004). What's in a translation rule? In *Proceedings of the North American Chapter of the Association for Computational Linguistics*
11740 (NAACL), pp. 273–280.
- 11742 Galley, M., K. R. McKeown, E. Fosler-Lussier, and H. Jing (2003). Discourse segmentation
11743 of multi-party conversation. In *Proceedings of the Association for Computational Linguistics*
11744 (ACL).
- 11745 Ganchev, K. and M. Dredze (2008). Small statistical models by random feature mixing. In *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, pp. 19–20.
- 11747 Ganchev, K., J. Graça, J. Gillenwater, and B. Taskar (2010). Posterior regularization for
11748 structured latent variable models. *The Journal of Machine Learning Research* 11, 2001–
11749 2049.
- 11750 Ganin, Y., E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand,
11751 and V. Lempitsky (2016). Domain-adversarial training of neural networks. *Journal of
11752 Machine Learning Research* 17(59), 1–35.
- 11753 Gao, J., G. Andrew, M. Johnson, and K. Toutanova (2007). A comparative study of param-
11754 eter estimation methods for statistical natural language processing. In *Proceedings of the
11755 Association for Computational Linguistics (ACL)*, pp. 824–831.
- 11756 Garg, N., L. Schiebinger, D. Jurafsky, and J. Zou (2018). Word embeddings quantify
11757 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sci-
11758 ences* 115(16), E3635–E3644.
- 11759 Gatt, A. and E. Krahmer (2018). Survey of the state of the art in natural language genera-
11760 tion: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research* 61,
11761 65–170.
- 11762 Ge, D., X. Jiang, and Y. Ye (2011). A note on the complexity of l p minimization. *Mathe-
11763 matical programming* 129(2), 285–299.
- 11764 Ge, N., J. Hale, and E. Charniak (1998). A statistical approach to anaphora resolution. In *Proceedings of the sixth workshop on very large corpora*, Volume 71, pp. 76.
- 11766 Ge, R., F. Huang, C. Jin, and Y. Yuan (2015). Escaping from saddle points — online stochas-
11767 tic gradient for tensor decomposition. In P. Grünwald, E. Hazan, and S. Kale (Eds.),
11768 *Proceedings of the Conference On Learning Theory (COLT)*.

- 11769 Ge, R. and R. J. Mooney (2005). A statistical semantic parser that integrates syntax and
 11770 semantics. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp.
 11771 9–16.
- 11772 Geach, P. T. (1962). *Reference and generality: An examination of some medieval and modern*
 11773 *theories*. Cornell University Press.
- 11774 Gehring, J., M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin (2017, May). Convolutional
 11775 sequence to sequence learning. *ArXiv e-prints*.
- 11776 Gildea, D. and D. Jurafsky (2002). Automatic labeling of semantic roles. *Computational*
 11777 *linguistics* 28(3), 245–288.
- 11778 Gimpel, K., N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yo-
 11779 gatama, J. Flanigan, and N. A. Smith (2011). Part-of-speech tagging for Twitter: an-
 11780 notation, features, and experiments. In *Proceedings of the Association for Computational*
 11781 *Linguistics (ACL)*, pp. 42–47.
- 11782 Glass, J., T. J. Hazen, S. Cyphers, I. Malioutov, D. Huynh, and R. Barzilay (2007). Recent
 11783 progress in the mit spoken lecture processing project. In *Eighth Annual Conference of the*
 11784 *International Speech Communication Association*.
- 11785 Glorot, X. and Y. Bengio (2010). Understanding the difficulty of training deep feedforward
 11786 neural networks. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*, pp. 249–
 11787 256.
- 11788 Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier networks. In *Proceedings*
 11789 *of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP*
 11790 *Volume*, Volume 15, pp. 315–323.
- 11791 Godfrey, J. J., E. C. Holliman, and J. McDaniel (1992). Switchboard: Telephone speech
 11792 corpus for research and development. In *Proceedings of the International Conference on*
 11793 *Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 517–520. IEEE.
- 11794 Goldberg, Y. (2017a, June). An adversarial review of “adversarial generation of
 11795 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)
 11796 [adversarial-generation-of-natural-language-409ac3378bd7](https://medium.com/@yoav.goldberg/an-adversarial-review-of-adversarial-generation-of-natural-language-409ac3378bd7).
- 11797 Goldberg, Y. (2017b). *Neural Network Methods for Natural Language Processing*. Synthesis
 11798 Lectures on Human Language Technologies. Morgan & Claypool Publishers.
- 11799 Goldberg, Y. and M. Elhadad (2010). An efficient algorithm for easy-first non-directional
 11800 dependency parsing. In *Proceedings of the North American Chapter of the Association for*
 11801 *Computational Linguistics (NAACL)*, pp. 742–750.

- 11802 Goldberg, Y. and J. Nivre (2012). A dynamic oracle for arc-eager dependency parsing.
11803 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
11804 959–976.
- 11805 Goldberg, Y., K. Zhao, and L. Huang (2013). Efficient implementation of beam-search
11806 incremental parsers. In *ACL (2)*, pp. 628–633.
- 11807 Goldwater, S. and T. Griffiths (2007). A fully bayesian approach to unsupervised part-of-
11808 speech tagging. In *Annual meeting-association for computational linguistics*, Volume 45.
- 11809 Gonçalo Oliveira, H. R., F. A. Cardoso, and F. C. Pereira (2007). Tra-la-lyrics: An approach
11810 to generate text based on rhythm. In *Proceedings of the 4th. International Joint Workshop*
11811 on *Computational Creativity*. A. Cardoso and G. Wiggins.
- 11812 Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT Press.
- 11813 Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Lan-*
11814 *guage* 15(4), 403–434.
- 11815 Gouws, S., D. Metzler, C. Cai, and E. Hovy (2011). Contextual bearing on linguistic varia-
11816 tion in social media. In *LASM*.
- 11817 Goyal, A., H. Daume III, and S. Venkatasubramanian (2009). Streaming for large scale
11818 nlp: Language modeling. In *Proceedings of the North American Chapter of the Association*
11819 for *Computational Linguistics (NAACL)*, pp. 512–520.
- 11820 Graves, A. (2012). Sequence transduction with recurrent neural networks. In *Proceedings*
11821 of the *International Conference on Machine Learning (ICML)*.
- 11822 Graves, A. and N. Jaitly (2014). Towards end-to-end speech recognition with recur-
11823 rent neural networks. In *Proceedings of the International Conference on Machine Learning*
11824 (*ICML*), pp. 1764–1772.
- 11825 Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirec-
11826 tional lstm and other neural network architectures. *Neural Networks* 18(5), 602–610.
- 11827 Grice, H. P. (1975). Logic and conversation. In P. Cole and J. L. Morgan (Eds.), *Syntax and*
11828 *Semantics Volume 3: Speech Acts*, pp. 41–58. Academic Press.
- 11829 Grishman, R. (2012). Information extraction: Capabilities and challenges. Notes prepared
11830 for the 2012 International Winter School in Language and Speech Technologies, Rovira
11831 i Virgili University, Tarragona, Spain.
- 11832 Grishman, R. (2015). Information extraction. *IEEE Intelligent Systems* 30(5), 8–15.

- 11833 Grishman, R., C. Macleod, and J. Sterling (1992). Evaluating parsing strategies using
11834 standardized parse files. In *Proceedings of the third conference on Applied natural language
11835 processing*, pp. 156–161. Association for Computational Linguistics.
- 11836 Grishman, R. and B. Sundheim (1996). Message understanding conference-6: A brief his-
11837 tory. In *Proceedings of the International Conference on Computational Linguistics (COLING)*,
11838 pp. 466–471.
- 11839 Groenendijk, J. and M. Stokhof (1991). Dynamic predicate logic. *Linguistics and philoso-*
11840 *phy* 14(1), 39–100.
- 11841 Grosz, B. J. (1979). Focusing and description in natural language dialogues. Technical
11842 report, SRI INTERNATIONAL MENLO PARK CA.
- 11843 Grosz, B. J., S. Weinstein, and A. K. Joshi (1995). Centering: A framework for modeling
11844 the local coherence of discourse. *Computational linguistics* 21(2), 203–225.
- 11845 Gu, J., Z. Lu, H. Li, and V. O. Li (2016). Incorporating copying mechanism in sequence-to-
11846 sequence learning. In *Proceedings of the Association for Computational Linguistics (ACL)*,
11847 pp. 1631–1640.
- 11848 Gulcehre, C., S. Ahn, R. Nallapati, B. Zhou, and Y. Bengio (2016). Pointing the unknown
11849 words. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 140–149.
- 11850 Gutmann, M. U. and A. Hyvärinen (2012). Noise-contrastive estimation of unnormalized
11851 statistical models, with applications to natural image statistics. *The Journal of Machine
11852 Learning Research* 13(1), 307–361.
- 11853 Haghghi, A. and D. Klein (2007). Unsupervised coreference resolution in a nonparametric
11854 bayesian model. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11855 Haghghi, A. and D. Klein (2009). Simple coreference resolution with rich syntactic and
11856 semantic features. In *Proceedings of Empirical Methods for Natural Language Processing
11857 (EMNLP)*, pp. 1152–1161.
- 11858 Haghghi, A. and D. Klein (2010). Coreference resolution in a modular, entity-centered
11859 model. In *Proceedings of the North American Chapter of the Association for Computational
11860 Linguistics (NAACL)*, pp. 385–393.
- 11861 Hajíč, J. and B. Hladká (1998). Tagging inflective languages: Prediction of morphological
11862 categories for a rich, structured tagset. In *Proceedings of the Association for Computational
11863 Linguistics (ACL)*, pp. 483–490.
- 11864 Halliday, M. and R. Hasan (1976). *Cohesion in English*. London: Longman.

- 11865 Hammerton, J. (2003). Named entity recognition with long short-term memory. In *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 172–175.
- 11866
- 11867 Han, X. and L. Sun (2012). An entity-topic model for entity linking. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 105–115.
- 11868
- 11869 Han, X., L. Sun, and J. Zhao (2011). Collective entity linking in web text: a graph-based method. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, pp. 765–774.
- 11870
- 11871
- 11872 Hannak, A., E. Anderson, L. F. Barrett, S. Lehmann, A. Mislove, and M. Riedewald (2012). Tweetin’ in the rain: Exploring societal-scale effects of weather on mood. In *Proceedings of the International Conference on Web and Social Media (ICWSM)*.
- 11873
- 11874
- 11875 Hardmeier, C. (2012). Discourse in statistical machine translation. a survey and a case study. *Discours. Revue de linguistique, psycholinguistique et informatique. A journal of linguistics, psycholinguistics and computational linguistics* (11).
- 11876
- 11877
- 11878 Haspelmath, M. and A. Sims (2013). *Understanding morphology*. Routledge.
- 11879
- 11880 Hastie, T., R. Tibshirani, and J. Friedman (2009). *The elements of statistical learning* (Second ed.). New York: Springer.
- 11881
- 11882 Hatzivassiloglou, V. and K. R. McKeown (1997). Predicting the semantic orientation of adjectives. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 174–181.
- 11883
- 11884 Hayes, A. F. and K. Krippendorff (2007). Answering the call for a standard reliability measure for coding data. *Communication methods and measures* 1(1), 77–89.
- 11885
- 11886 He, H., A. Balakrishnan, M. Eric, and P. Liang (2017). Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1766–1776.
- 11887
- 11888
- 11889 He, K., X. Zhang, S. Ren, and J. Sun (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 1026–1034.
- 11890
- 11891
- 11892 He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 770–778.
- 11893
- 11894 He, L., K. Lee, M. Lewis, and L. Zettlemoyer (2017). Deep semantic role labeling: What works and what’s next. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 11895
- 11896

- 11897 He, Z., S. Liu, M. Li, M. Zhou, L. Zhang, and H. Wang (2013). Learning entity representation for entity disambiguation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 30–34.
- 11900 Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 539–545. Association for Computational Linguistics.
- 11903 Hearst, M. A. (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational linguistics* 23(1), 33–64.
- 11905 Heerschap, B., F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, and F. de Jong (2011). 11906 Polarity analysis of texts using discourse structure. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 1061–1070. ACM.
- 11908 Henderson, J. (2004). Discriminative training of a neural network statistical parser. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 95–102.
- 11910 Hendrickx, I., S. N. Kim, Z. Kozareva, P. Nakov, D. Ó Séaghdha, S. Padó, M. Pennacchiotti, 11911 L. Romano, and S. Szpakowicz (2009). Semeval-2010 task 8: Multi-way classification of 11912 semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, pp. 94–99. Association for Com- 11913 putational Linguistics.
- 11915 Hermann, K. M., T. Kočiský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and 11916 P. Blunsom (2015). Teaching machines to read and comprehend. In *Advances in Neu- 11917 ral Information Processing Systems*, pp. 1693–1701.
- 11918 Hernault, H., H. Prendinger, D. A. duVerle, and M. Ishizuka (2010). HILDA: A discourse 11919 parser using support vector machine classification. *Dialogue and Discourse* 1(3), 1–33.
- 11920 Hill, F., A. Bordes, S. Chopra, and J. Weston (2016). The goldilocks principle: Reading 11921 children’s books with explicit memory representations. In *Proceedings of the International 11922 Conference on Learning Representations (ICLR)*.
- 11923 Hill, F., K. Cho, and A. Korhonen (2016). Learning distributed representations of sentences 11924 from unlabelled data. In *Proceedings of the North American Chapter of the Association for 11925 Computational Linguistics (NAACL)*.
- 11926 Hill, F., R. Reichart, and A. Korhonen (2015). Simlex-999: Evaluating semantic models 11927 with (genuine) similarity estimation. *Computational Linguistics* 41(4), 665–695.
- 11928 Hindle, D. and M. Rooth (1993). Structural ambiguity and lexical relations. *Computational 11929 linguistics* 19(1), 103–120.

- 11930 Hirao, T., Y. Yoshida, M. Nishino, N. Yasuda, and M. Nagata (2013). Single-document
11931 summarization as a tree knapsack problem. In *Proceedings of Empirical Methods for Nat-*
11932 *ural Language Processing (EMNLP)*, pp. 1515–1520.
- 11933 Hirschman, L. and R. Gaizauskas (2001). Natural language question answering: the view
11934 from here. *natural language engineering* 7(4), 275–300.
- 11935 Hirschman, L., M. Light, E. Breck, and J. D. Burger (1999). Deep read: A reading compre-
11936 hension system. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11937 325–332.
- 11938 Hobbs, J. R. (1978). Resolving pronoun references. *Lingua* 44(4), 311–338.
- 11939 Hobbs, J. R., D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson (1997).
11940 Fastus: A cascaded finite-state transducer for extracting information from natural-
11941 language text. *Finite-state language processing*, 383–406.
- 11942 Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. *Neural computa-*
11943 *tion* 9(8), 1735–1780.
- 11944 Hockenmaier, J. and M. Steedman (2007). Ccgbank: a corpus of ccg derivations and de-
11945 pendency structures extracted from the penn treebank. *Computational Linguistics* 33(3),
11946 355–396.
- 11947 Hoffart, J., M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva,
11948 S. Thater, and G. Weikum (2011). Robust disambiguation of named entities in text. In
11949 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 782–792.
- 11950 Hoffmann, R., C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld (2011). Knowledge-based
11951 weak supervision for information extraction of overlapping relations. In *Proceedings of*
11952 *the Association for Computational Linguistics (ACL)*, pp. 541–550.
- 11953 Holmstrom, L. and P. Koistinen (1992). Using additive noise in back-propagation training.
11954 *IEEE Transactions on Neural Networks* 3(1), 24–38.
- 11955 Hovy, D., S. Spruit, M. Mitchell, E. M. Bender, M. Strube, and H. Wallach (2017). Proceed-
11956 ings of the first acl workshop on ethics in natural language processing. In *Proceedings of*
11957 *the First ACL Workshop on Ethics in Natural Language Processing*. Association for Compu-
11958 tational Linguistics.
- 11959 Hovy, E. and J. Lavid (2010). Towards a ‘science’ of corpus annotation: a new method-
11960 ological challenge for corpus linguistics. *International journal of translation* 22(1), 13–36.
- 11961 Hsu, D., S. M. Kakade, and T. Zhang (2012). A spectral algorithm for learning hidden
11962 markov models. *Journal of Computer and System Sciences* 78(5), 1460–1480.

- 11963 Hu, M. and B. Liu (2004). Mining and summarizing customer reviews. In *Proceedings of
11964 Knowledge Discovery and Data Mining (KDD)*, pp. 168–177.
- 11965 Hu, Z., Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing (2017). Toward controlled
11966 generation of text. In *International Conference on Machine Learning*, pp. 1587–1596.
- 11967 Huang, F. and A. Yates (2012). Biased representation learning for domain adaptation. In
11968 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1313–1323.
- 11969 Huang, L. and D. Chiang (2007). Forest rescoring: Faster decoding with integrated lan-
11970 guage models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
11971 144–151.
- 11972 Huang, L., S. Fayong, and Y. Guo (2012). Structured perceptron with inexact search. In
11973 *Proceedings of the North American Chapter of the Association for Computational Linguistics
(NAACL)*, pp. 142–151.
- 11975 Huang, Y. (2015). *Pragmatics* (Second ed.). Oxford Textbooks in Linguistics. Oxford Uni-
11976 versity Press.
- 11977 Huang, Z., W. Xu, and K. Yu (2015). Bidirectional lstm-crf models for sequence tagging.
11978 *arXiv preprint arXiv:1508.01991*.
- 11979 Huddleston, R. and G. K. Pullum (2005). *A student's introduction to English grammar*. Cam-
11980 bridge University Press.
- 11981 Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes.
11982 *Proceedings of the IRE* 40(9), 1098–1101.
- 11983 Humphreys, K., R. Gaizauskas, and S. Azzam (1997). Event coreference for information
11984 extraction. In *Proceedings of a Workshop on Operational Factors in Practical, Robust Anaphora
11985 Resolution for Unrestricted Texts*, pp. 75–81. Association for Computational Linguistics.
- 11986 Ide, N. and Y. Wilks (2006). Making sense about sense. In *Word sense disambiguation*, pp.
11987 47–73. Springer.
- 11988 Ioffe, S. and C. Szegedy (2015). Batch normalization: Accelerating deep network train-
11989 ing by reducing internal covariate shift. In *Proceedings of the International Conference on
11990 Machine Learning (ICML)*, pp. 448–456.
- 11991 Isozaki, H., T. Hirao, K. Duh, K. Sudoh, and H. Tsukada (2010). Automatic evaluation
11992 of translation quality for distant language pairs. In *Proceedings of Empirical Methods for
11993 Natural Language Processing (EMNLP)*, pp. 944–952.

- 11994 Ivanova, A., S. Oepen, L. Øvrelid, and D. Flickinger (2012, July). Who did what to whom?
11995 a contrastive study of syntacto-semantic dependencies. In *Proceedings of the Sixth Lin-*
11996 *guistic Annotation Workshop*, Jeju, Republic of Korea, pp. 2–11. Association for Compu-
11997 tational Linguistics.
- 11998 Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. Daumé III (2015). Deep unordered com-
11999 position rivals syntactic methods for text classification. In *Proceedings of the Association*
12000 *for Computational Linguistics (ACL)*, pp. 1681–1691.
- 12001 James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learn-*
12002 *ing*, Volume 112. Springer.
- 12003 Janin, A., D. Baron, J. Edwards, D. Ellis, D. Gelbart, N. Morgan, B. Peskin, T. Pfau,
12004 E. Shriberg, A. Stolcke, et al. (2003). The ICSI meeting corpus. In *Acoustics, Speech,*
12005 *and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference*
12006 *on*, Volume 1, pp. I–I. IEEE.
- 12007 Jean, S., K. Cho, R. Memisevic, and Y. Bengio (2015). On using very large target vocab-
12008 uary for neural machine translation. In *Proceedings of the Association for Computational*
12009 *Linguistics (ACL)*, pp. 1–10.
- 12010 Jeong, M., C.-Y. Lin, and G. G. Lee (2009). Semi-supervised speech act recognition in
12011 emails and forums. In *Proceedings of Empirical Methods for Natural Language Processing*
12012 (*EMNLP*), pp. 1250–1259.
- 12013 Ji, H. and R. Grishman (2011). Knowledge base population: Successful approaches and
12014 challenges. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1148–
12015 1158.
- 12016 Ji, Y., T. Cohn, L. Kong, C. Dyer, and J. Eisenstein (2015). Document context language
12017 models. In *International Conference on Learning Representations, Workshop Track*, Volume
12018 abs/1511.03962.
- 12019 Ji, Y. and J. Eisenstein (2014). Representation learning for text-level discourse parsing. In
12020 *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12021 Ji, Y. and J. Eisenstein (2015, June). One vector is not enough: Entity-augmented distribu-
12022 tional semantics for discourse relations. *Transactions of the Association for Computational*
12023 *Linguistics (TACL)*.
- 12024 Ji, Y., G. Haffari, and J. Eisenstein (2016). A latent variable recurrent neural network for
12025 discourse relation language models. In *Proceedings of the North American Chapter of the*
12026 *Association for Computational Linguistics (NAACL)*.

- 12027 Ji, Y. and N. A. Smith (2017). Neural discourse structure for text categorization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 996–1005.
- 12029 Ji, Y., C. Tan, S. Martschat, Y. Choi, and N. A. Smith (2017). Dynamic entity representations
12030 in neural language models. In *Proceedings of Empirical Methods for Natural Language
12031 Processing (EMNLP)*, pp. 1831–1840.
- 12032 Jiang, L., M. Yu, M. Zhou, X. Liu, and T. Zhao (2011). Target-dependent twitter sentiment
12033 classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
12034 151–160.
- 12035 Jing, H. (2000). Sentence reduction for automatic text summarization. In *Proceedings of
12036 the sixth conference on Applied natural language processing*, pp. 310–315. Association for
12037 Computational Linguistics.
- 12038 Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of
12039 Knowledge Discovery and Data Mining (KDD)*, pp. 133–142.
- 12040 Jockers, M. L. (2015). Szuzhet? <http://bla.bla.com>.
- 12041 Johnson, A. E., T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody,
12042 P. Szolovits, L. A. Celi, and R. G. Mark (2016). Mimic-iii, a freely accessible critical care
12043 database. *Scientific data* 3, 160035.
- 12044 Johnson, M. (1998). Pcfg models of linguistic tree representations. *Computational Linguistics*
12045 24(4), 613–632.
- 12046 Johnson, R. and T. Zhang (2017). Deep pyramid convolutional neural networks for text
12047 categorization. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
12048 562–570.
- 12049 Joshi, A. K. (1985). How much context-sensitivity is required to provide reasonable struc-
12050 tural descriptions? – tree adjoining grammars. In *Natural Language Processing – Theoret-
12051 ical, Computational and Psychological Perspective*. New York, NY: Cambridge University
12052 Press.
- 12053 Joshi, A. K. and Y. Schabes (1997). Tree-adjoining grammars. In *Handbook of formal lan-
12054 guages*, pp. 69–123. Springer.
- 12055 Joshi, A. K., K. V. Shanker, and D. Weir (1991). The convergence of mildly context-sensitive
12056 grammar formalisms. In *Foundational Issues in Natural Language Processing*. Cambridge
12057 MA: MIT Press.
- 12058 Jozefowicz, R., O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu (2016). Exploring the limits
12059 of language modeling. *arXiv preprint arXiv:1602.02410*.

- 12060 Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). An empirical exploration of recurrent
12061 network architectures. In *Proceedings of the International Conference on Machine Learning*
12062 (*ICML*), pp. 2342–2350.
- 12063 Jurafsky, D. (1996). A probabilistic model of lexical and syntactic access and disambiguation.
12064 *Cognitive Science* 20(2), 137–194.
- 12065 Jurafsky, D. and J. H. Martin (2009). *Speech and Language Processing* (Second ed.). Prentice
12066 Hall.
- 12067 Jurafsky, D. and J. H. Martin (2018). *Speech and Language Processing* (Third ed.). Prentice
12068 Hall.
- 12069 Kadlec, R., M. Schmid, O. Bajgar, and J. Kleindienst (2016). Text understanding with
12070 the attention sum reader network. In *Proceedings of the Association for Computational*
12071 *Linguistics (ACL)*, pp. 908–918.
- 12072 Kalchbrenner, N. and P. Blunsom (2013, August). Recurrent convolutional neural net-
12073 works for discourse compositionality. In *Proceedings of the Workshop on Continuous Vec-
12074 tor Space Models and their Compositionality*, Sofia, Bulgaria, pp. 119–126. Association for
12075 Computational Linguistics.
- 12076 Kalchbrenner, N., L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu
12077 (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.
- 12078 Kalchbrenner, N., E. Grefenstette, and P. Blunsom (2014). A convolutional neural network
12079 for modelling sentences. In *Proceedings of the Association for Computational Linguistics*
12080 (*ACL*), pp. 655–665.
- 12081 Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Lin-
12082 guistics* 43(02), 365–392.
- 12083 Kate, R. J., Y. W. Wong, and R. J. Mooney (2005). Learning to transform natural to formal
12084 languages. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 12085 Kehler, A. (2007). Rethinking the SMASH approach to pronoun interpretation. In *Interdis-
12086 ciplinary perspectives on reference processing*, New Directions in Cognitive Science Series,
12087 pp. 95–122. Oxford University Press.
- 12088 Kibble, R. and R. Power (2004). Optimizing referential coherence in text generation. *Com-
12089 putational Linguistics* 30(4), 401–416.
- 12090 Kilgarriff, A. (1997). I don't believe in word senses. *Computers and the Humanities* 31(2),
12091 91–113.

- 12092 Kilgarriff, A. and G. Grefenstette (2003). Introduction to the special issue on the web as
 12093 corpus. *Computational linguistics* 29(3), 333–347.
- 12094 Kim, M.-J. (2002). Does korean have adjectives? *MIT Working Papers in Linguistics* 43,
 12095 71–89.
- 12096 Kim, S.-M. and E. Hovy (2006, July). Extracting opinions, opinion holders, and topics
 12097 expressed in online news media text. In *Proceedings of the Workshop on Sentiment and*
 12098 *Subjectivity in Text*, Sydney, Australia, pp. 1–8. Association for Computational Linguis-
 12099 tics.
- 12100 Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings*
 12101 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1746–1751.
- 12102 Kim, Y., C. Denton, L. Hoang, and A. M. Rush (2017). Structured attention networks. In
 12103 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12104 Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush (2016). Character-aware neural language
 12105 models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- 12106 Kingma, D. and J. Ba (2014). Adam: A method for stochastic optimization. *arXiv preprint*
 12107 *arXiv:1412.6980*.
- 12108 Kiperwasser, E. and Y. Goldberg (2016). Simple and accurate dependency parsing using
 12109 bidirectional lstm feature representations. *Transactions of the Association for Compu-
 12110 tational Linguistics* 4, 313–327.
- 12111 Kipper-Schuler, K. (2005). *VerbNet: A broad-coverage, comprehensive verb lexicon*. Ph. D.
 12112 thesis, Computer and Information Science, University of Pennsylvania.
- 12113 Kiros, R., R. Salakhutdinov, and R. Zemel (2014). Multimodal neural language models. In
 12114 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 595–603.
- 12115 Kiros, R., Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler
 12116 (2015). Skip-thought vectors. In *Neural Information Processing Systems (NIPS)*.
- 12117 Klein, D. and C. D. Manning (2003). Accurate unlexicalized parsing. In *Proceedings of the*
 12118 *Association for Computational Linguistics (ACL)*, pp. 423–430.
- 12119 Klein, D. and C. D. Manning (2004). Corpus-based induction of syntactic structure: Mod-
 12120 els of dependency and constituency. In *Proceedings of the Association for Computational*
 12121 *Linguistics (ACL)*.
- 12122 Klein, G., Y. Kim, Y. Deng, J. Senellart, and A. M. Rush (2017). Opennmt: Open-source
 12123 toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

- 12124 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.
- 12127 Klenner, M. (2007). Enforcing consistency on coreference sets. In *Recent Advances in Natural Language Processing (RANLP)*, pp. 323–328.
- 12129 Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics* 25(4), 607–615.
- 12131 Knight, K. and J. Graehl (1998). Machine transliteration. *Computational Linguistics* 24(4), 599–612.
- 12133 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.
- 12136 Knight, K. and J. May (2009). Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pp. 571–596. Springer.
- 12138 Knott, A. (1996). *A data-driven methodology for motivating a set of coherence relations*. Ph. D. thesis, The University of Edinburgh.
- 12140 Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, Volume 5, pp. 79–86.
- 12142 Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.
- 12143 Koehn, P. (2017). Neural machine translation. *arXiv preprint arXiv:1709.07809*.
- 12144 Konstas, I. and M. Lapata (2013). A global model for concept-to-text generation. *Journal of Artificial Intelligence Research* 48, 305–346.
- 12146 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.
- 12149 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.
- 12151 Koo, T. and M. Collins (2010). Efficient third-order dependency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12153 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.

- 12156 Kovach, B. and T. Rosenstiel (2014). *The elements of journalism: What newspeople should know
12157 and the public should expect*. Three Rivers Press.
- 12158 Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon. In
12159 *Proceedings of the North American Chapter of the Association for Computational Linguistics
12160 (NAACL)*, pp. 606–616.
- 12161 Krishnamurthy, J. and T. M. Mitchell (2012). Weakly supervised training of semantic
12162 parsers. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
12163 pp. 754–765.
- 12164 Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep
12165 convolutional neural networks. In *Neural Information Processing Systems (NIPS)*, pp.
12166 1097–1105.
- 12167 Kübler, S., R. McDonald, and J. Nivre (2009). Dependency parsing. *Synthesis Lectures on
12168 Human Language Technologies* 1(1), 1–127.
- 12169 Kuhlmann, M. and J. Nivre (2010). Transition-based techniques for non-projective depen-
12170 dency parsing. *Northern European Journal of Language Technology (NEJLT)* 2(1), 1–19.
- 12171 Kummerfeld, J. K., T. Berg-Kirkpatrick, and D. Klein (2015). An empirical analysis of op-
12172 timization for max-margin NLP. In *Proceedings of Empirical Methods for Natural Language
12173 Processing (EMNLP)*.
- 12174 Kwiatkowski, T., S. Goldwater, L. Zettlemoyer, and M. Steedman (2012). A probabilistic
12175 model of syntactic and semantic acquisition from child-directed utterances and their
12176 meanings. In *Proceedings of the European Chapter of the Association for Computational Lin-
12177 guistics (EACL)*, pp. 234–244.
- 12178 Lafferty, J., A. McCallum, and F. Pereira (2001). Conditional random fields: Probabilistic
12179 models for segmenting and labeling sequence data. In *Proceedings of the International
12180 Conference on Machine Learning (ICML)*.
- 12181 Lakoff, G. (1973). Hedges: A study in meaning criteria and the logic of fuzzy concepts.
12182 *Journal of philosophical logic* 2(4), 458–508.
- 12183 Lample, G., M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer (2016). Neural
12184 architectures for named entity recognition. In *Proceedings of the North American Chapter
12185 of the Association for Computational Linguistics (NAACL)*, pp. 260–270.
- 12186 Langkilde, I. and K. Knight (1998). Generation that exploits corpus-based statistical
12187 knowledge. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 704–
12188 710.

- 12189 Lapata, M. (2003). Probabilistic text structuring: Experiments with sentence ordering. In
12190 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 545–552.
- 12191 Lappin, S. and H. J. Leass (1994). An algorithm for pronominal anaphora resolution.
12192 *Computational linguistics* 20(4), 535–561.
- 12193 Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using
12194 the inside-outside algorithm. *Computer speech & language* 4(1), 35–56.
- 12195 Lascarides, A. and N. Asher (2007). Segmented discourse representation theory: Dynamic
12196 semantics with discourse structure. In *Computing meaning*, pp. 87–124. Springer.
- 12197 Law, E. and L. v. Ahn (2011). Human computation. *Synthesis Lectures on Artificial Intelli-*
12198 *gence and Machine Learning* 5(3), 1–121.
- 12199 Lebret, R., D. Grangier, and M. Auli (2016). Neural text generation from structured data
12200 with application to the biography domain. In *Proceedings of Empirical Methods for Natural*
12201 *Language Processing (EMNLP)*, pp. 1203–1213.
- 12202 LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural*
12203 *networks: Tricks of the trade*, pp. 9–50. Springer.
- 12204 Lee, C. M. and S. S. Narayanan (2005). Toward detecting emotions in spoken dialogs.
12205 *IEEE transactions on speech and audio processing* 13(2), 293–303.
- 12206 Lee, H., A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky (2013). De-
12207 terministic coreference resolution based on entity-centric, precision-ranked rules. *Com-*
12208 *putational Linguistics* 39(4), 885–916.
- 12209 Lee, H., Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky (2011). Stan-
12210 ford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In
12211 *Proceedings of the Conference on Natural Language Learning (CoNLL)*, pp. 28–34. Associa-
12212 tion for Computational Linguistics.
- 12213 Lee, K., L. He, M. Lewis, and L. Zettlemoyer (2017). End-to-end neural coreference reso-
12214 lution. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12215 Lee, K., L. He, and L. Zettlemoyer (2018). Higher-order coreference resolution with coarse-
12216 to-fine inference. In *Proceedings of the North American Chapter of the Association for Com-*
12217 *putational Linguistics (NAACL)*, pp. 687–692.
- 12218 Lenat, D. B., R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd (1990). Cyc: toward
12219 programs with common sense. *Communications of the ACM* 33(8), 30–49.

- 12220 Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries:
12221 how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual interna-*
12222 *tional conference on Systems documentation*, pp. 24–26. ACM.
- 12223 Levesque, H. J., E. Davis, and L. Morgenstern (2011). The Winograd schema challenge. In
12224 *AAAI spring symposium: Logical formalizations of commonsense reasoning*, Volume 46, pp.
12225 47.
- 12226 Levin, E., R. Pieraccini, and W. Eckert (1998). Using markov decision process for learning
12227 dialogue strategies. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the*
12228 *1998 IEEE International Conference on*, Volume 1, pp. 201–204. IEEE.
- 12229 Levy, O. and Y. Goldberg (2014). Dependency-based word embeddings. In *Proceedings of*
12230 *the Association for Computational Linguistics (ACL)*, pp. 302–308.
- 12231 Levy, O., Y. Goldberg, and I. Dagan (2015). Improving distributional similarity with
12232 lessons learned from word embeddings. *Transactions of the Association for Computational*
12233 *Linguistics* 3, 211–225.
- 12234 Levy, R. and C. Manning (2009). An informal introduction to computational semantics.
- 12235 Lewis, M. and M. Steedman (2013). Combined distributional and logical semantics. *Trans-*
12236 *actions of the Association for Computational Linguistics* 1, 179–192.
- 12237 Lewis II, P. M. and R. E. Stearns (1968). Syntax-directed transduction. *Journal of the ACM*
12238 (*JACM*) 15(3), 465–488.
- 12239 Li, J. and D. Jurafsky (2015). Do multi-sense embeddings improve natural language
12240 understanding? In *Proceedings of Empirical Methods for Natural Language Processing*
12241 (*EMNLP*), pp. 1722–1732.
- 12242 Li, J. and D. Jurafsky (2017). Neural net models of open-domain discourse coherence. In
12243 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 198–209.
- 12244 Li, J., R. Li, and E. Hovy (2014). Recursive deep models for discourse parsing. In *Proceed-*
12245 *ings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12246 Li, J., M.-T. Luong, and D. Jurafsky (2015). A hierarchical neural autoencoder for para-
12247 graphs and documents. In *Proceedings of Empirical Methods for Natural Language Process-*
12248 *ing (EMNLP)*.
- 12249 Li, J., T. Luong, D. Jurafsky, and E. Hovy (2015). When are tree structures necessary
12250 for deep learning of representations? In *Proceedings of Empirical Methods for Natural*
12251 *Language Processing (EMNLP)*, pp. 2304–2314.

- 12252 Li, J., W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao (2016). Deep reinforcement
12253 learning for dialogue generation. In *Proceedings of Empirical Methods for Natural Language
12254 Processing (EMNLP)*, pp. 1192–1202.
- 12255 Li, Q., S. Anzaroot, W.-P. Lin, X. Li, and H. Ji (2011). Joint inference for cross-document
12256 information extraction. In *Proceedings of the International Conference on Information and
12257 Knowledge Management (CIKM)*, pp. 2225–2228.
- 12258 Li, Q., H. Ji, and L. Huang (2013). Joint event extraction via structured prediction with
12259 global features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
12260 73–82.
- 12261 Liang, P. (2005). *Semi-supervised learning for natural language*. Ph. D. thesis, Massachusetts
12262 Institute of Technology.
- 12263 Liang, P., A. Bouchard-Côté, D. Klein, and B. Taskar (2006). An end-to-end discriminative
12264 approach to machine translation. In *Proceedings of the Association for Computational
12265 Linguistics (ACL)*, pp. 761–768.
- 12266 Liang, P., M. Jordan, and D. Klein (2009). Learning semantic correspondences with less
12267 supervision. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–
12268 99.
- 12269 Liang, P., M. I. Jordan, and D. Klein (2013). Learning dependency-based compositional
12270 semantics. *Computational Linguistics* 39(2), 389–446.
- 12271 Liang, P. and D. Klein (2009). Online EM for unsupervised models. In *Proceedings of
12272 the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp.
12273 611–619.
- 12274 Liang, P., S. Petrov, M. I. Jordan, and D. Klein (2007). The infinite pcfg using hierarchical
12275 dirichlet processes. In *Proceedings of Empirical Methods for Natural Language Processing
12276 (EMNLP)*, pp. 688–697.
- 12277 Liang, P. and C. Potts (2015). Bringing machine learning and compositional semantics
12278 together. *Annual Review of Linguistics* 1(1), 355–376.
- 12279 Lieber, R. (2015). *Introducing morphology*. Cambridge University Press.
- 12280 Lin, D. (1998). Automatic retrieval and clustering of similar words. In *Proceedings of the
12281 International Conference on Computational Linguistics (COLING)*, pp. 768–774.
- 12282 Lin, J. and C. Dyer (2010). Data-intensive text processing with mapreduce. *Synthesis
12283 Lectures on Human Language Technologies* 3(1), 1–177.

- 12284 Lin, Z., M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio (2017). A struc-
12285 tured self-attentive sentence embedding. In *Proceedings of the International Conference on*
12286 *Learning Representations (ICLR)*.
- 12287 Lin, Z., M.-Y. Kan, and H. T. Ng (2009). Recognizing implicit discourse relations in the
12288 penn discourse treebank. In *Proceedings of Empirical Methods for Natural Language Pro-*
12289 *cessing (EMNLP)*, pp. 343–351.
- 12290 Lin, Z., H. T. Ng, and M.-Y. Kan (2011). Automatically evaluating text coherence using
12291 discourse relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12292 pp. 997–1006.
- 12293 Lin, Z., H. T. Ng, and M.-Y. Kan (2014). A pdtb-styled end-to-end discourse parser. *Natural*
12294 *Language Engineering* 20(2), 151–184.
- 12295 Ling, W., C. Dyer, A. Black, and I. Trancoso (2015). Two/too simple adaptations of
12296 word2vec for syntax problems. In *Proceedings of the North American Chapter of the As-*
12297 *sociation for Computational Linguistics (NAACL)*.
- 12298 Ling, W., T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso
12299 (2015). Finding function in form: Compositional character models for open vocabulary
12300 word representation. In *Proceedings of Empirical Methods for Natural Language Processing*
12301 (*EMNLP*).
- 12302 Ling, W., G. Xiang, C. Dyer, A. Black, and I. Trancoso (2013). Microblogs as parallel cor-
12303 pora. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12304 Ling, X., S. Singh, and D. S. Weld (2015). Design challenges for entity linking. *Transactions*
12305 *of the Association for Computational Linguistics* 3, 315–328.
- 12306 Linguistic Data Consortium (2005, July). ACE (automatic content extraction) English an-
12307 notation guidelines for relations. Technical Report Version 5.8.3, Linguistic Data Con-
12308 sortium.
- 12309 Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge
12310 University Press.
- 12311 Liu, D. C. and J. Nocedal (1989). On the limited memory BFGS method for large scale
12312 optimization. *Mathematical programming* 45(1-3), 503–528.
- 12313 Liu, Y., Q. Liu, and S. Lin (2006). Tree-to-string alignment template for statistical machine
12314 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 609–
12315 616.

- 12316 Loper, E. and S. Bird (2002). NLTK: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics*, pp. 63–70. Association for Computational Linguistics.
- 12319 Louis, A., A. Joshi, and A. Nenkova (2010). Discourse indicators for content selection in summarization. In *Proceedings of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, pp. 147–156.
- 12322 Louis, A. and A. Nenkova (2013). What makes writing great? first experiments on article quality prediction in the science journalism domain. *Transactions of the Association for Computational Linguistics* 1, 341–352.
- 12325 Loveland, D. W. (2016). *Automated Theorem Proving: a logical basis*. Elsevier.
- 12326 Lowe, R., N. Pow, I. V. Serban, and J. Pineau (2015). The Ubuntu Dialogue Corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*.
- 12329 Luo, X. (2005). On coreference resolution performance metrics. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 25–32.
- 12331 Luo, X., A. Ittycheriah, H. Jing, N. Kambhatla, and S. Roukos (2004). A mention-synchronous coreference resolution algorithm based on the bell tree. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12334 Luong, M.-T., R. Socher, and C. D. Manning (2013). Better word representations with recursive neural networks for morphology. *CoNLL-2013* 104.
- 12336 Luong, T., H. Pham, and C. D. Manning (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1412–1421.
- 12339 Luong, T., I. Sutskever, Q. Le, O. Vinyals, and W. Zaremba (2015). Addressing the rare word problem in neural machine translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 11–19.
- 12342 Maas, A. L., A. Y. Hannun, and A. Y. Ng (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the International Conference on Machine Learning (ICML)*.
- 12345 Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 276–283.
- 12347 Mairesse, F. and M. A. Walker (2011). Controlling user perceptions of linguistic style: Trainable generation of personality traits. *Computational Linguistics* 37(3), 455–488.

- 12349 Mani, I., M. Verhagen, B. Wellner, C. M. Lee, and J. Pustejovsky (2006). Machine learning
 12350 of temporal relations. In *Proceedings of the Association for Computational Linguistics (ACL)*,
 12351 pp. 753–760.
- 12352 Mann, W. C. and S. A. Thompson (1988). Rhetorical structure theory: Toward a functional
 12353 theory of text organization. *Text* 8(3), 243–281.
- 12354 Manning, C. D. (2015). Last words: Computational linguistics and deep learning. *Computa-
 12355 tional Linguistics* 41(4), 701–707.
- 12356 Manning, C. D., P. Raghavan, H. Schütze, et al. (2008). *Introduction to information retrieval*,
 12357 Volume 1. Cambridge university press.
- 12358 Manning, C. D. and H. Schütze (1999). *Foundations of Statistical Natural Language Process-
 12359 ing*. Cambridge, Massachusetts: MIT press.
- 12360 Marcu, D. (1996). Building up rhetorical structure trees. In *Proceedings of the National
 12361 Conference on Artificial Intelligence*, pp. 1069–1074.
- 12362 Marcu, D. (1997a). From discourse structures to text summaries. In *Proceedings of the
 12363 workshop on Intelligent Scalable Text Summarization*.
- 12364 Marcu, D. (1997b). From local to global coherence: A bottom-up approach to text plan-
 12365 ning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 629–635.
- 12366 Marcus, M. P., M. A. Marcinkiewicz, and B. Santorini (1993). Building a large annotated
 12367 corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330.
- 12368 Maron, O. and T. Lozano-Pérez (1998). A framework for multiple-instance learning. In
 12369 *Neural Information Processing Systems (NIPS)*, pp. 570–576.
- 12370 Márquez, G. G. (1970). *One Hundred Years of Solitude*. Harper & Row. English translation
 12371 by Gregory Rabassa.
- 12372 Martins, A. F. T., N. A. Smith, and E. P. Xing (2009). Concise integer linear programming
 12373 formulations for dependency parsing. In *Proceedings of the Association for Computational
 12374 Linguistics (ACL)*, pp. 342–350.
- 12375 Martins, A. F. T., N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo (2010).
 12376 Turbo parsers: Dependency parsing by approximate variational inference. In *Proceed-
 12377 ings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 34–44.
- 12378 Matsuzaki, T., Y. Miyao, and J. Tsujii (2005). Probabilistic cfg with latent annotations. In
 12379 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 75–82.

- 12380 Matthiessen, C. and J. A. Bateman (1991). *Text generation and systemic-functional linguistics: experiences from English and Japanese*. Pinter Publishers.
- 12381
- 12382 McCallum, A. and W. Li (2003). Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 188–191.
- 12383
- 12384
- 12385
- 12386 McCallum, A. and B. Wellner (2004). Conditional models of identity uncertainty with application to noun coreference. In *NIPS*, pp. 905–912.
- 12387
- 12388 McDonald, R., K. Crammer, and F. Pereira (2005). Online large-margin training of dependency parsers. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 91–98.
- 12389
- 12390
- 12391 McDonald, R., K. Hannan, T. Neylon, M. Wells, and J. Reynar (2007). Structured models for fine-to-coarse sentiment analysis. In *Proceedings of ACL*.
- 12392
- 12393 McDonald, R. and F. Pereira (2006). Online learning of approximate dependency parsing algorithms. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)*.
- 12394
- 12395
- 12396 McKeown, K. (1992). *Text generation*. Cambridge University Press.
- 12397
- 12398
- 12399
- 12400 McKeown, K., S. Rosenthal, K. Thadani, and C. Moore (2010). Time-efficient creation of an accurate sentence fusion corpus. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 317–320.
- 12401
- 12402
- 12403
- 12404
- 12405
- 12406
- 12407
- 12408
- 12409
- 12410
- 12411
- McNamee, P. and H. T. Dang (2009). Overview of the tac 2009 knowledge base population track. In *Text Analysis Conference (TAC)*, Volume 17, pp. 111–113.
- Medlock, B. and T. Briscoe (2007). Weakly supervised learning for hedge classification in scientific literature. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 992–999.
- Mei, H., M. Bansal, and M. R. Walter (2016). What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 720–730.

- 12412 Merity, S., N. S. Keskar, and R. Socher (2018). Regularizing and optimizing lstm language
12413 models. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12414 Merity, S., C. Xiong, J. Bradbury, and R. Socher (2017). Pointer sentinel mixture models.
12415 In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 12416 Messud, C. (2014, June). A new 'l'étranger'. *New York Review of Books*.
- 12417 Miao, Y. and P. Blunsom (2016). Language as a latent variable: Discrete generative mod-
12418 els for sentence compression. In *Proceedings of Empirical Methods for Natural Language
12419 Processing (EMNLP)*, pp. 319–328.
- 12420 Miao, Y., L. Yu, and P. Blunsom (2016). Neural variational inference for text processing. In
12421 *Proceedings of the International Conference on Machine Learning (ICML)*.
- 12422 Mihalcea, R., T. A. Chklovski, and A. Kilgarriff (2004, July). The senseval-3 english lexical
12423 sample task. In *Proceedings of SENSEVAL-3*, Barcelona, Spain, pp. 25–28. Association for
12424 Computational Linguistics.
- 12425 Mihalcea, R. and D. Radev (2011). *Graph-based natural language processing and information
12426 retrieval*. Cambridge University Press.
- 12427 Mikolov, T., K. Chen, G. Corrado, and J. Dean (2013). Efficient estimation of word repre-
12428 sentations in vector space. In *Proceedings of International Conference on Learning Represen-
12429 tations*.
- 12430 Mikolov, T., A. Deoras, D. Povey, L. Burget, and J. Cernocky (2011). Strategies for train-
12431 ing large scale neural network language models. In *Automatic Speech Recognition and
12432 Understanding (ASRU), 2011 IEEE Workshop on*, pp. 196–201. IEEE.
- 12433 Mikolov, T., M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur (2010). Recurrent
12434 neural network based language model. In *INTERSPEECH*, pp. 1045–1048.
- 12435 Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). Distributed rep-
12436 resentations of words and phrases and their compositionality. In *Advances in Neural
12437 Information Processing Systems*, pp. 3111–3119.
- 12438 Mikolov, T., W.-t. Yih, and G. Zweig (2013). Linguistic regularities in continuous space
12439 word representations. In *Proceedings of the North American Chapter of the Association for
12440 Computational Linguistics (NAACL)*, pp. 746–751.
- 12441 Mikolov, T. and G. Zweig. Context dependent recurrent neural network language model.
12442 In *Proceedings of Spoken Language Technology (SLT)*, pp. 234–239.
- 12443 Miller, G. A., G. A. Heise, and W. Lichten (1951). The intelligibility of speech as a function
12444 of the context of the test materials. *Journal of experimental psychology* 41(5), 329.

- 12445 Miller, M., C. Sathi, D. Wiesenthal, J. Leskovec, and C. Potts (2011). Sentiment flow
12446 through hyperlink networks. In *Proceedings of the International Conference on Web and*
12447 *Social Media (ICWSM)*.
- 12448 Miller, S., J. Guinness, and A. Zamanian (2004). Name tagging with word clusters and
12449 discriminative training. In *Proceedings of the North American Chapter of the Association for*
12450 *Computational Linguistics (NAACL)*, pp. 337–342.
- 12451 Milne, D. and I. H. Witten (2008). Learning to link with wikipedia. In *Proceedings of the*
12452 *International Conference on Information and Knowledge Management (CIKM)*, pp. 509–518.
12453 ACM.
- 12454 Miltsakaki, E. and K. Kukich (2004). Evaluation of text coherence for electronic essay
12455 scoring systems. *Natural Language Engineering* 10(1), 25–55.
- 12456 Minka, T. P. (1999). From hidden markov models to linear dynamical systems. Tech. Rep.
12457 531, Vision and Modeling Group of Media Lab, MIT.
- 12458 Minsky, M. (1974). A framework for representing knowledge. Technical Report 306, MIT
12459 AI Laboratory.
- 12460 Minsky, M. and S. Papert (1969). *Perceptrons*. MIT press.
- 12461 Mintz, M., S. Bills, R. Snow, and D. Jurafsky (2009). Distant supervision for relation extrac-
12462 tion without labeled data. In *Proceedings of the Association for Computational Linguistics*
12463 (*ACL*), pp. 1003–1011.
- 12464 Mirza, P., R. Sprugnoli, S. Tonelli, and M. Speranza (2014). Annotating causality in the
12465 tempeval-3 corpus. In *Proceedings of the EACL 2014 Workshop on Computational Ap-*
12466 *proaches to Causality in Language (CAtoCL)*, pp. 10–19.
- 12467 Misra, D. K. and Y. Artzi (2016). Neural shift-reduce ccg semantic parsing. In *Proceedings*
12468 *of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12469 Mitchell, J. and M. Lapata (2010). Composition in distributional models of semantics.
12470 *Cognitive Science* 34(8), 1388–1429.
- 12471 Miwa, M. and M. Bansal (2016). End-to-end relation extraction using lstms on sequences
12472 and tree structures. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12473 pp. 1105–1116.
- 12474 Mnih, A. and G. Hinton (2007). Three new graphical models for statistical language mod-
12475 ellng. In *Proceedings of the 24th international conference on Machine learning, ICML '07*,
12476 New York, NY, USA, pp. 641–648. ACM.

- 12477 Mnih, A. and G. E. Hinton (2008). A scalable hierarchical distributed language model. In
12478 *Neural Information Processing Systems (NIPS)*, pp. 1081–1088.
- 12479 Mnih, A. and Y. W. Teh (2012). A fast and simple algorithm for training neural probabilis-
12480 tic language models. In *Proceedings of the International Conference on Machine Learning*
12481 (*ICML*).
- 12482 Mohammad, S. M. and P. D. Turney (2013). Crowdsourcing a word–emotion association
12483 lexicon. *Computational Intelligence* 29(3), 436–465.
- 12484 Mohri, M., F. Pereira, and M. Riley (2002). Weighted finite-state transducers in speech
12485 recognition. *Computer Speech & Language* 16(1), 69–88.
- 12486 Mohri, M., A. Rostamizadeh, and A. Talwalkar (2012). *Foundations of machine learning*.
12487 MIT press.
- 12488 Montague, R. (1973). The proper treatment of quantification in ordinary english. In *Ap-
12489 proaches to natural language*, pp. 221–242. Springer.
- 12490 Moore, J. D. and C. L. Paris (1993, dec). Planning text for advisory dialogues: Capturing
12491 intentional and rhetorical information. *Comput. Linguist.* 19(4), 651–694.
- 12492 Morante, R. and E. Blanco (2012). *sem 2012 shared task: Resolving the scope and fo-
12493 cus of negation. In *Proceedings of the First Joint Conference on Lexical and Computational
12494 Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2:
12495 Proceedings of the Sixth International Workshop on Semantic Evaluation*, pp. 265–274. Asso-
12496 ciation for Computational Linguistics.
- 12497 Morante, R. and W. Daelemans (2009). Learning the scope of hedge cues in biomedical
12498 texts. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language
12499 Processing*, pp. 28–36. Association for Computational Linguistics.
- 12500 Morante, R. and C. Sporleder (2012). Modality and negation: An introduction to the
12501 special issue. *Computational linguistics* 38(2), 223–260.
- 12502 Mostafazadeh, N., A. Grealish, N. Chambers, J. Allen, and L. Vanderwende (2016, June).
12503 Caters: Causal and temporal relation scheme for semantic annotation of event struc-
12504 tures. In *Proceedings of the Fourth Workshop on Events*, San Diego, California, pp. 51–61.
12505 Association for Computational Linguistics.
- 12506 Mueller, T., H. Schmid, and H. Schütze (2013). Efficient higher-order CRFs for morpholog-
12507 ical tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*,
12508 pp. 322–332.

- 12509 Müller, C. and M. Strube (2006). Multi-level annotation of linguistic data with mmax2.
12510 *Corpus technology and language pedagogy: New resources, new tools, new methods 3*, 197–
12511 214.
- 12512 Muralidharan, A. and M. A. Hearst (2013). Supporting exploratory text analysis in litera-
12513 ture study. *Literary and linguistic computing* 28(2), 283–295.
- 12514 Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press.
- 12515 Nakagawa, T., K. Inui, and S. Kurohashi (2010). Dependency tree-based sentiment classi-
12516 fication using crfs with hidden variables. In *Proceedings of the North American Chapter of*
12517 *the Association for Computational Linguistics (NAACL)*, pp. 786–794.
- 12518 Nakazawa, T., M. Yaguchi, K. Uchimoto, M. Utayama, E. Sumita, S. Kurohashi, and H. Isa-
12519 hara (2016). ASPEC: Asian scientific paper excerpt corpus. In *Proceedings of the Language*
12520 *Resources and Evaluation Conference*, pp. 2204–2208.
- 12521 Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys*
12522 (CSUR) 41(2), 10.
- 12523 Neal, R. M. and G. E. Hinton (1998). A view of the em algorithm that justifies incremental,
12524 sparse, and other variants. In *Learning in graphical models*, pp. 355–368. Springer.
- 12525 Nenkova, A. and K. McKeown (2012). A survey of text summarization techniques. In
12526 *Mining text data*, pp. 43–76. Springer.
- 12527 Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tu-
12528 torial. *arXiv preprint arXiv:1703.01619*.
- 12529 Neubig, G., C. Dyer, Y. Goldberg, A. Matthews, W. Ammar, A. Anastasopoulos, M. Balles-
12530 teros, D. Chiang, D. Clothiaux, T. Cohn, K. Duh, M. Faruqui, C. Gan, D. Garrette,
12531 Y. Ji, L. Kong, A. Kuncoro, G. Kumar, C. Malaviya, P. Michel, Y. Oda, M. Richardson,
12532 N. Saphra, S. Swayamdipta, and P. Yin (2017). Dynet: The dynamic neural network
12533 toolkit.
- 12534 Neubig, G., Y. Goldberg, and C. Dyer (2017). On-the-fly operation batching in dynamic
12535 computation graphs. In *Neural Information Processing Systems (NIPS)*.
- 12536 Neubig, G., M. Sperber, X. Wang, M. Felix, A. Matthews, S. Padmanabhan, Y. Qi, D. S.
12537 Sachan, P. Arthur, P. Godard, J. Hewitt, R. Riad, and L. Wang (2018, March). XNMT:
12538 The extensible neural machine translation toolkit. In *Conference of the Association for*
12539 *Machine Translation in the Americas (AMTA) Open Source Software Showcase*, Boston.
- 12540 Neuhaus, P. and N. Bröker (1997). The complexity of recognition of linguistically adequate
12541 dependency grammars. In *eacl*, pp. 337–343.

- 12542 Newman, D., C. Chemudugunta, and P. Smyth (2006). Statistical entity-topic models. In
 12543 *Proceedings of Knowledge Discovery and Data Mining (KDD)*, pp. 680–686.
- 12544 Ng, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In
 12545 *Proceedings of the 48th annual meeting of the association for computational linguistics*, pp.
 12546 1396–1411. Association for Computational Linguistics.
- 12547 Nguyen, D. and A. S. Dogruöz (2013). Word level language identification in online multi-
 12548 lingual communication. In *Proceedings of Empirical Methods for Natural Language Process-*
 12549 *ing (EMNLP)*.
- 12550 Nguyen, D. T. and S. Joty (2017). A neural local coherence model. In *Proceedings of the*
 12551 *Association for Computational Linguistics (ACL)*, pp. 1320–1330.
- 12552 Nigam, K., A. K. McCallum, S. Thrun, and T. Mitchell (2000). Text classification from
 12553 labeled and unlabeled documents using em. *Machine learning* 39(2-3), 103–134.
- 12554 Nirenburg, S. and Y. Wilks (2001). What's in a symbol: ontology, representation and lan-
 12555 guage. *Journal of Experimental & Theoretical Artificial Intelligence* 13(1), 9–23.
- 12556 Nivre, J. (2008). Algorithms for deterministic incremental dependency parsing. *Compu-*
 12557 *tational Linguistics* 34(4), 513–553.
- 12558 Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald,
 12559 S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman (2016, may). Universal de-
 12560 pendencies v1: A multilingual treebank collection. In N. C. C. Chair, K. Choukri, T. De-
 12561 clerck, S. Goggi, M. Grobelnik, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk,
 12562 and S. Piperidis (Eds.), *Proceedings of the Tenth International Conference on Language Re-*
 12563 *sources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Asso-
 12564 *ciation (ELRA)*.
- 12565 Nivre, J. and J. Nilsson (2005). Pseudo-projective dependency parsing. In *Proceedings of the*
 12566 *43rd Annual Meeting on Association for Computational Linguistics*, pp. 99–106. Association
 12567 for Computational Linguistics.
- 12568 Novikoff, A. B. (1962). On convergence proofs on perceptrons. In *Proceedings of the Sym-*
 12569 *posium on the Mathematical Theory of Automata*, Volume 12, pp. 615—622.
- 12570 Och, F. J. and H. Ney (2003). A systematic comparison of various statistical alignment
 12571 models. *Computational linguistics* 29(1), 19–51.
- 12572 O'Connor, B., M. Krieger, and D. Ahn (2010). Tweetmotif: Exploratory search and topic
 12573 summarization for twitter. In *Proceedings of the International Conference on Web and Social*
 12574 *Media (ICWSM)*, pp. 384–385.

- 12575 Oepen, S., M. Kuhlmann, Y. Miyao, D. Zeman, D. Flickinger, J. Hajic, A. Ivanova, and
12576 Y. Zhang (2014). SemEval 2014 task 8: Broad-coverage semantic dependency parsing.
12577 In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pp.
12578 63–72.
- 12579 Oflazer, K. and İ. Kuruöz (1994). Tagging and morphological disambiguation of turkish
12580 text. In *Proceedings of the fourth conference on Applied natural language processing*, pp. 144–
12581 149. Association for Computational Linguistics.
- 12582 Ohta, T., Y. Tateisi, and J.-D. Kim (2002). The genia corpus: An annotated research abstract
12583 corpus in molecular biology domain. In *Proceedings of the second international conference*
12584 on *Human Language Technology Research*, pp. 82–86. Morgan Kaufmann Publishers Inc.
- 12585 Onishi, T., H. Wang, M. Bansal, K. Gimpel, and D. McAllester (2016). Who did what: A
12586 large-scale person-centered cloze dataset. In *Proceedings of Empirical Methods for Natural*
12587 *Language Processing (EMNLP)*, pp. 2230–2235.
- 12588 Owoputi, O., B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith (2013).
12589 Improved part-of-speech tagging for online conversational text with word clusters. In
12590 *Proceedings of the North American Chapter of the Association for Computational Linguistics*
12591 (NAACL), pp. 380–390.
- 12592 Packard, W., E. M. Bender, J. Read, S. Oepen, and R. Dridan (2014). Simple negation
12593 scope resolution through deep parsing: A semantic solution to a semantic problem. In
12594 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 69–78.
- 12595 Paice, C. D. (1990). Another stemmer. In *ACM SIGIR Forum*, Volume 24, pp. 56–61.
- 12596 Pak, A. and P. Paroubek (2010). Twitter as a corpus for sentiment analysis and opinion
12597 mining. In *LREC*, Volume 10, pp. 1320–1326.
- 12598 Palmer, F. R. (2001). *Mood and modality*. Cambridge University Press.
- 12599 Palmer, M., D. Gildea, and P. Kingsbury (2005). The proposition bank: An annotated
12600 corpus of semantic roles. *Computational linguistics* 31(1), 71–106.
- 12601 Pan, S. J. and Q. Yang (2010). A survey on transfer learning. *IEEE Transactions on knowledge*
12602 *and data engineering* 22(10), 1345–1359.
- 12603 Pang, B. and L. Lee (2004). A sentimental education: Sentiment analysis using subjectivity
12604 summarization based on minimum cuts. In *Proceedings of the Association for Compu-*
12605 *tational Linguistics (ACL)*, pp. 271–278.
- 12606 Pang, B. and L. Lee (2005). Seeing stars: Exploiting class relationships for sentiment cate-
12607 gorization with respect to rating scales. In *Proceedings of the Association for Computational*
12608 *Linguistics (ACL)*, pp. 115–124.

- 12609 Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. *Foundations and trends
12610 in information retrieval* 2(1-2), 1–135.
- 12611 Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up?: sentiment classification using
12612 machine learning techniques. In *Proceedings of Empirical Methods for Natural Language
12613 Processing (EMNLP)*, pp. 79–86.
- 12614 Papineni, K., S. Roukos, T. Ward, and W.-J. Zhu (2002). Bleu: a method for automatic
12615 evaluation of machine translation. In *Proceedings of the Association for Computational
12616 Linguistics (ACL)*, pp. 311–318.
- 12617 Park, J. and C. Cardie (2012). Improving implicit discourse relation recognition through
12618 feature set optimization. In *Proceedings of the Special Interest Group on Discourse and Dia-
12619 logue (SIGDIAL)*, pp. 108–112.
- 12620 Parsons, T. (1990). *Events in the Semantics of English*, Volume 5. MIT Press.
- 12621 Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural
12622 networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pp.
12623 1310–1318.
- 12624 Paul, M., M. Federico, and S. Stüker (2010). Overview of the iwslt 2010 evaluation cam-
12625 paign. In *International Workshop on Spoken Language Translation (IWSLT) 2010*.
- 12626 Pedersen, T., S. Patwardhan, and J. Michelizzi (2004). Wordnet::similarity - measuring the
12627 relatedness of concepts. In *Proceedings of the North American Chapter of the Association for
12628 Computational Linguistics (NAACL)*, pp. 38–41.
- 12629 Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blon-
12630 del, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau,
12631 M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in
12632 Python. *Journal of Machine Learning Research* 12, 2825–2830.
- 12633 Pei, W., T. Ge, and B. Chang (2015). An effective neural network model for graph-based
12634 dependency parsing. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12635 pp. 313–322.
- 12636 Peldszus, A. and M. Stede (2013). From argument diagrams to argumentation mining
12637 in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence
12638 (IJCINI)* 7(1), 1–31.
- 12639 Peldszus, A. and M. Stede (2015). An annotated corpus of argumentative microtexts. In
12640 *Proceedings of the First Conference on Argumentation*.

- 12641 Peng, F., F. Feng, and A. McCallum (2004). Chinese segmentation and new word detection using conditional random fields. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 562.
- 12644 Pennington, J., R. Socher, and C. Manning (2014). Glove: Global vectors for word representation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1532–1543.
- 12647 Pereira, F. and Y. Schabes (1992). Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 128–135.
- 12650 Pereira, F. C. N. and S. M. Shieber (2002). *Prolog and natural-language analysis*. Microtome Publishing.
- 12652 Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer (2018). Deep contextualized word representations. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*.
- 12655 Peterson, W. W., T. G. Birdsall, and W. C. Fox (1954). The theory of signal detectability. *Transactions of the IRE professional group on information theory* 4(4), 171–212.
- 12657 Petrov, S., L. Barrett, R. Thibaux, and D. Klein (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12660 Petrov, S., D. Das, and R. McDonald (2012, May). A universal part-of-speech tagset. In *Proceedings of LREC*.
- 12662 Petrov, S. and R. McDonald (2012). Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, Volume 59.
- 12665 Pinker, S. (2003). *The language instinct: How the mind creates language*. Penguin UK.
- 12666 Pinter, Y., R. Guthrie, and J. Eisenstein (2017). Mimicking word embeddings using subword RNNs. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12669 Pitler, E., A. Louis, and A. Nenkova (2009). Automatic sense prediction for implicit discourse relations in text. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12672 Pitler, E. and A. Nenkova (2009). Using syntax to disambiguate explicit discourse connectives in text. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 13–16.

- 12675 Pitler, E., M. Raghupathy, H. Mehta, A. Nenkova, A. Lee, and A. Joshi (2008). Easily iden-
 12676 tifiable discourse relations. In *Proceedings of the International Conference on Computational*
 12677 *Linguistics (COLING)*, pp. 87–90.
- 12678 Plank, B., A. Søgaard, and Y. Goldberg (2016). Multilingual part-of-speech tagging with
 12679 bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the*
 12680 *Association for Computational Linguistics (ACL)*.
- 12681 Poesio, M., R. Stevenson, B. Di Eugenio, and J. Hitzeman (2004). Centering: A parametric
 12682 theory and its instantiations. *Computational linguistics* 30(3), 309–363.
- 12683 Polanyi, L. and A. Zaenen (2006). Contextual valence shifters. In *Computing attitude and*
 12684 *affect in text: Theory and applications*. Springer.
- 12685 Ponzetto, S. P. and M. Strube (2006). Exploiting semantic role labeling, wordnet and
 12686 wikipedia for coreference resolution. In *Proceedings of the North American Chapter of*
 12687 *the Association for Computational Linguistics (NAACL)*, pp. 192–199.
- 12688 Ponzetto, S. P. and M. Strube (2007). Knowledge derived from wikipedia for computing
 12689 semantic relatedness. *Journal of Artificial Intelligence Research* 30, 181–212.
- 12690 Poon, H. and P. Domingos (2008). Joint unsupervised coreference resolution with markov
 12691 logic. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
 12692 650–659.
- 12693 Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of Em-*
 12694 *pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–10.
- 12695 Popel, M., D. Marecek, J. Stepánek, D. Zeman, and Z. Zabokrtský (2013). Coordination
 12696 structures in dependency treebanks. In *Proceedings of the Association for Computational*
 12697 *Linguistics (ACL)*, pp. 517–527.
- 12698 Popescu, A.-M., O. Etzioni, and H. Kautz (2003). Towards a theory of natural language
 12699 interfaces to databases. In *Proceedings of Intelligent User Interfaces (IUI)*, pp. 149–157.
- 12700 Poplack, S. (1980). Sometimes i'll start a sentence in spanish y termino en español: toward
 12701 a typology of code-switching1. *Linguistics* 18(7-8), 581–618.
- 12702 Porter, M. F. (1980). An algorithm for suffix stripping. *Program* 14(3), 130–137.
- 12703 Prabhakaran, V., O. Rambow, and M. Diab (2010). Automatic committed belief tagging.
 12704 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
 12705 1014–1022.

- 12706 Pradhan, S., X. Luo, M. Recasens, E. Hovy, V. Ng, and M. Strube (2014). Scoring coreference partitions of predicted mentions: A reference implementation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 30–35.
- 12709 Pradhan, S., L. Ramshaw, M. Marcus, M. Palmer, R. Weischedel, and N. Xue (2011). CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pp. 1–27. Association for Computational Linguistics.
- 12713 Pradhan, S., W. Ward, K. Hacioglu, J. H. Martin, and D. Jurafsky (2005). Semantic role labeling using different syntactic views. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 581–588.
- 12716 Prasad, R., N. Dinesh, A. Lee, E. Miltsakaki, L. Robaldo, A. Joshi, and B. Webber (2008). The Penn Discourse Treebank 2.0. In *Proceedings of LREC*.
- 12718 Punyakanok, V., D. Roth, and W.-t. Yih (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics* 34(2), 257–287.
- 12720 Pustejovsky, J., P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, et al. (2003). The timebank corpus. In *Corpus linguistics*, Volume 2003, pp. 40. Lancaster, UK.
- 12723 Pustejovsky, J., B. Ingria, R. Sauri, J. Castano, J. Littman, R. Gaizauskas, A. Setzer, G. Katz, and I. Mani (2005). The specification language timeml. In *The language of time: A reader*, pp. 545–557. Oxford University Press.
- 12726 Qin, L., Z. Zhang, H. Zhao, Z. Hu, and E. Xing (2017). Adversarial connective-exploiting networks for implicit discourse relation classification. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1006–1017.
- 12729 Qiu, G., B. Liu, J. Bu, and C. Chen (2011). Opinion word expansion and target extraction through double propagation. *Computational linguistics* 37(1), 9–27.
- 12731 Quattoni, A., S. Wang, L.-P. Morency, M. Collins, and T. Darrell (2007). Hidden conditional random fields. *IEEE transactions on pattern analysis and machine intelligence* 29(10).
- 12733 Rahman, A. and V. Ng (2011). Narrowing the modeling gap: a cluster-ranking approach to coreference resolution. *Journal of Artificial Intelligence Research* 40, 469–521.
- 12735 Rajpurkar, P., J. Zhang, K. Lopyrev, and P. Liang (2016). Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2383–2392.

- 12738 Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2016). Sequence level training with
 12739 recurrent neural networks. In *Proceedings of the International Conference on Learning Rep-*
 12740 *resentations (ICLR)*.
- 12741 Rao, D., D. Yarowsky, A. Shreevats, and M. Gupta (2010). Classifying latent user attributes
 12742 in twitter. In *Proceedings of Workshop on Search and mining user-generated contents*.
- 12743 Ratinov, L. and D. Roth (2009). Design challenges and misconceptions in named entity
 12744 recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language*
 12745 *Learning*, pp. 147–155. Association for Computational Linguistics.
- 12746 Ratinov, L., D. Roth, D. Downey, and M. Anderson (2011). Local and global algorithms
 12747 for disambiguation to wikipedia. In *Proceedings of the Association for Computational Lin-*
 12748 *guistics (ACL)*, pp. 1375–1384.
- 12749 Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2007). (approximate) subgradient methods
 12750 for structured prediction. In *Proceedings of Artificial Intelligence and Statistics (AISTATS)*,
 12751 pp. 380–387.
- 12752 Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *emnlp*,
 12753 pp. 133–142.
- 12754 Ratnaparkhi, A., J. Reynar, and S. Roukos (1994). A maximum entropy model for preposi-
 12755 tional phrase attachment. In *Proceedings of the workshop on Human Language Technology*,
 12756 pp. 250–255. Association for Computational Linguistics.
- 12757 Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for
 12758 sentiment classification. In *Proceedings of the ACL student research workshop*, pp. 43–48.
 12759 Association for Computational Linguistics.
- 12760 Reisinger, D., R. Rudinger, F. Ferraro, C. Harman, K. Rawlins, and B. V. Durme (2015).
 12761 Semantic proto-roles. *Transactions of the Association for Computational Linguistics* 3, 475–
 12762 488.
- 12763 Reisinger, J. and R. J. Mooney (2010). Multi-prototype vector-space models of word mean-
 12764 ing. In *Proceedings of the North American Chapter of the Association for Computational Lin-*
 12765 *guistics (NAACL)*, pp. 109–117.
- 12766 Reiter, E. and R. Dale (2000). *Building natural language generation systems*. Cambridge
 12767 university press.
- 12768 Resnik, P., M. B. Olsen, and M. Diab (1999). The bible as a parallel corpus: Annotating the
 12769 ‘book of 2000 tongues’. *Computers and the Humanities* 33(1-2), 129–153.

- 12770 Resnik, P. and N. A. Smith (2003). The web as a parallel corpus. *Computational Linguistics* 29(3), 349–380.
- 12771
- 12772 Ribeiro, F. N., M. Araújo, P. Gonçalves, M. A. Gonçalves, and F. Benevenuto (2016).
12773 Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis meth-
12774 ods. *EPJ Data Science* 5(1), 1–29.
- 12775 Richardson, M., C. J. Burges, and E. Renshaw (2013). MCTest: A challenge dataset for
12776 the open-domain machine comprehension of text. In *Proceedings of Empirical Methods for*
12777 *Natural Language Processing (EMNLP)*, pp. 193–203.
- 12778 Riedel, S., L. Yao, and A. McCallum (2010). Modeling relations and their mentions without
12779 labeled text. In *Proceedings of the European Conference on Machine Learning and Principles*
12780 *and Practice of Knowledge Discovery in Databases (ECML)*, pp. 148–163.
- 12781 Riedl, M. O. and R. M. Young (2010). Narrative planning: Balancing plot and character.
12782 *Journal of Artificial Intelligence Research* 39, 217–268.
- 12783 Rieser, V. and O. Lemon (2011). *Reinforcement learning for adaptive dialogue systems: a data-
12784 driven methodology for dialogue management and natural language generation*. Springer Sci-
12785 ence & Business Media.
- 12786 Riloff, E. (1996). Automatically generating extraction patterns from untagged text. In
12787 *Proceedings of the national conference on artificial intelligence*, pp. 1044–1049.
- 12788 Riloff, E. and J. Wiebe (2003). Learning extraction patterns for subjective expressions. In
12789 *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pp.
12790 105–112. Association for Computational Linguistics.
- 12791 Ritchie, G. (2001). Current directions in computational humour. *Artificial Intelligence Re-
12792 view* 16(2), 119–135.
- 12793 Ritter, A., C. Cherry, and W. B. Dolan (2011). Data-driven response generation in social
12794 media. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp.
12795 583–593.
- 12796 Ritter, A., S. Clark, Mausam, and O. Etzioni (2011). Named entity recognition in tweets:
12797 an experimental study. In *Proceedings of EMNLP*.
- 12798 Roark, B., M. Saraclar, and M. Collins (2007). Discriminative $i_1 \dots i_n / i_1 \dots i_n$ -gram language
12799 modeling. *Computer Speech & Language* 21(2), 373–392.
- 12800 Robert, C. and G. Casella (2013). *Monte Carlo statistical methods*. Springer Science & Busi-
12801 ness Media.

- 12802 Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language mod-
12803 elling. *Computer Speech & Language* 10(3), 187–228.
- 12804 Ross, S., G. Gordon, and D. Bagnell (2011). A reduction of imitation learning and struc-
12805 tured prediction to no-regret online learning. In *Proceedings of Artificial Intelligence and*
12806 *Statistics (AISTATS)*, pp. 627–635.
- 12807 Roy, N., J. Pineau, and S. Thrun (2000). Spoken dialogue management using probabilistic
12808 reasoning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 93–
12809 100.
- 12810 Rudinger, R., J. Naradowsky, B. Leonard, and B. Van Durme (2018). Gender bias in coref-
12811 erence resolution. In *Proceedings of the North American Chapter of the Association for Com-*
12812 *putational Linguistics (NAACL)*.
- 12813 Rudnicky, A. and W. Xu (1999). An agenda-based dialog management architecture for
12814 spoken language systems. In *IEEE Automatic Speech Recognition and Understanding Work-*
12815 *shop*, Volume 13.
- 12816 Rush, A. M., S. Chopra, and J. Weston (2015). A neural attention model for abstractive sen-
12817 tence summarization. In *Proceedings of Empirical Methods for Natural Language Processing*
12818 (*EMNLP*), pp. 379–389.
- 12819 Rush, A. M., D. Sontag, M. Collins, and T. Jaakkola (2010). On dual decomposition and
12820 linear programming relaxations for natural language processing. In *Proceedings of Em-*
12821 *pirical Methods for Natural Language Processing (EMNLP)*, pp. 1–11.
- 12822 Russell, S. J. and P. Norvig (2009). *Artificial intelligence: a modern approach* (3rd ed.). Prentice
12823 Hall.
- 12824 Rutherford, A., V. Demberg, and N. Xue (2017). A systematic study of neural discourse
12825 models for implicit discourse relation. In *Proceedings of the European Chapter of the Asso-*
12826 *ciation for Computational Linguistics (EACL)*, pp. 281–291.
- 12827 Rutherford, A. T. and N. Xue (2014). Discovering implicit discourse relations through
12828 brown cluster pair representation and coreference patterns. In *Proceedings of the Euro-*
12829 *pean Chapter of the Association for Computational Linguistics (EACL)*.
- 12830 Sag, I. A., T. Baldwin, F. Bond, A. Copestake, and D. Flickinger (2002). Multiword expres-
12831 sions: A pain in the neck for nlp. In *International Conference on Intelligent Text Processing*
12832 *and Computational Linguistics*, pp. 1–15. Springer.
- 12833 Sagae, K. (2009). Analysis of discourse structure with syntactic dependencies and data-
12834 driven shift-reduce parsing. In *Proceedings of the 11th International Conference on Parsing*
12835 *Technologies*, pp. 81–84.

- 12836 Santos, C. D. and B. Zadrozny (2014). Learning character-level representations for part-of-
12837 speech tagging. In *Proceedings of the International Conference on Machine Learning (ICML)*,
12838 pp. 1818–1826.
- 12839 Sato, M.-A. and S. Ishii (2000). On-line em algorithm for the normalized gaussian network.
12840 *Neural computation* 12(2), 407–432.
- 12841 Saurí, R. and J. Pustejovsky (2009). Factbank: a corpus annotated with event factuality.
12842 *Language resources and evaluation* 43(3), 227.
- 12843 Saxe, A. M., J. L. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear
12844 dynamics of learning in deep linear neural networks. In *Proceedings of the International
12845 Conference on Learning Representations (ICLR)*.
- 12846 Schank, R. C. and R. Abelson (1977). *Scripts, goals, plans, and understanding*. Hillsdale, NJ:
12847 Erlbaum.
- 12848 Schapire, R. E. and Y. Singer (2000). Boostexter: A boosting-based system for text catego-
12849 rization. *Machine learning* 39(2-3), 135–168.
- 12850 Schaul, T., S. Zhang, and Y. LeCun (2013). No more pesky learning rates. In *Proceedings of
12851 the International Conference on Machine Learning (ICML)*, pp. 343–351.
- 12852 Schnabel, T., I. Labutov, D. Mimno, and T. Joachims (2015). Evaluation methods for un-
12853 supervised word embeddings. In *Proceedings of Empirical Methods for Natural Language
12854 Processing (EMNLP)*, pp. 298–307.
- 12855 Schneider, N., J. Flanigan, and T. O’Gorman (2015). The logic of amr: Practical, unified,
12856 graph-based sentence semantics for nlp. In *Proceedings of the North American Chapter of
12857 the Association for Computational Linguistics (NAACL)*, pp. 4–5.
- 12858 Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics* 24(1),
12859 97–123.
- 12860 Schwarm, S. E. and M. Ostendorf (2005). Reading level assessment using support vector
12861 machines and statistical language models. In *Proceedings of the Association for Compu-
12862 tational Linguistics (ACL)*, pp. 523–530.
- 12863 See, A., P. J. Liu, and C. D. Manning (2017). Get to the point: Summarization with pointer-
12864 generator networks. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12865 pp. 1073–1083.
- 12866 Sennrich, R., B. Haddow, and A. Birch (2016). Neural machine translation of rare words
12867 with subword units. In *Proceedings of the Association for Computational Linguistics (ACL)*,
12868 pp. 1715–1725.

- 12869 Serban, I. V., A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau (2016). Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 3776–3784.
- 12872 Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(1), 1–114.
- 12874 Shang, L., Z. Lu, and H. Li (2015). Neural responding machine for short-text conversation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1577–1586.
- 12876 Shen, D. and M. Lapata (2007). Using semantic roles to improve question answering. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 12–21.
- 12878 Shen, S., Y. Cheng, Z. He, W. He, H. Wu, M. Sun, and Y. Liu (2016). Minimum risk training for neural machine translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1683–1692.
- 12881 Shen, W., J. Wang, and J. Han (2015). Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27(2), 443–460.
- 12884 Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy* 8(3), 333–343.
- 12886 Siegelmann, H. T. and E. D. Sontag (1995). On the computational power of neural nets. *Journal of computer and system sciences* 50(1), 132–150.
- 12888 Singh, S., A. Subramanya, F. Pereira, and A. McCallum (2011). Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 793–803.
- 12891 Sipser, M. (2012). *Introduction to the Theory of Computation*. Cengage Learning.
- 12892 Smith, D. A. and J. Eisner (2006). Minimum risk annealing for training log-linear models. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 787–794.
- 12894 Smith, D. A. and J. Eisner (2008). Dependency parsing by belief propagation. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 145–156.
- 12896 Smith, D. A. and N. A. Smith (2007). Probabilistic models of nonprojective dependency trees. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 132–140.
- 12899 Smith, N. A. (2011). Linguistic structure prediction. *Synthesis Lectures on Human Language Technologies* 4(2), 1–274.

- 12901 Snover, M., B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul (2006). A study of transla-
12902 tion edit rate with targeted human annotation. In *Proceedings of association for machine*
12903 *translation in the Americas*, Volume 200.
- 12904 Snow, R., B. O'Connor, D. Jurafsky, and A. Y. Ng (2008). Cheap and fast—but is it good?:
12905 evaluating non-expert annotations for natural language tasks. In *Proceedings of Empirical*
12906 *Methods for Natural Language Processing (EMNLP)*, pp. 254–263.
- 12907 Snyder, B. and R. Barzilay (2007). Database-text alignment via structured multilabel classi-
12908 fication. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*,
12909 pp. 1713–1718.
- 12910 Socher, R., J. Bauer, C. D. Manning, and A. Y. Ng (2013). Parsing with compositional vector
12911 grammars. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12912 Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic compositionality
12913 through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Em-
12914 pirical Methods in Natural Language Processing and Computational Natural Language Learn-
12915 ing*, pp. 1201–1211. Association for Computational Linguistics.
- 12916 Socher, R., A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts (2013).
12917 Recursive deep models for semantic compositionality over a sentiment treebank. In
12918 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12919 Søgaard, A. (2013). Semi-supervised learning and domain adaptation in natural language
12920 processing. *Synthesis Lectures on Human Language Technologies* 6(2), 1–103.
- 12921 Solorio, T. and Y. Liu (2008). Learning to predict code-switching points. In *Proceedings*
12922 *of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 973–981. Association
12923 for Computational Linguistics.
- 12924 Somasundaran, S., G. Namata, J. Wiebe, and L. Getoor (2009). Supervised and unsuper-
12925 vised methods in employing discourse relations for improving opinion polarity classi-
12926 fication. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 12927 Somasundaran, S. and J. Wiebe (2009). Recognizing stances in online debates. In *Proceed-
12928 ings of the Association for Computational Linguistics (ACL)*, pp. 226–234.
- 12929 Song, L., B. Boots, S. M. Siddiqi, G. J. Gordon, and A. J. Smola (2010). Hilbert space
12930 embeddings of hidden markov models. In *Proceedings of the International Conference on*
12931 *Machine Learning (ICML)*, pp. 991–998.
- 12932 Song, L., Y. Zhang, X. Peng, Z. Wang, and D. Gildea (2016). Amr-to-text generation as
12933 a traveling salesman problem. In *Proceedings of Empirical Methods for Natural Language*
12934 *Processing (EMNLP)*, pp. 2084–2089.

- 12935 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.
- 12936
- 12937 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)*.
- 12938
- 12939
- 12940
- 12941 Soricut, 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.
- 12942
- 12943
- 12944 Sowa, J. F. (2000). *Knowledge representation: logical, philosophical, and computational foundations*. Pacific Grove, CA: Brooks/Cole.
- 12945
- 12946 Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28(1), 11–21.
- 12947
- 12948 Spitkovsky, V. I., H. Alshawi, D. Jurafsky, and C. D. Manning (2010). Viterbi training improves unsupervised dependency parsing. In *CONLL*, pp. 9–17.
- 12949
- 12950 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.
- 12951
- 12952
- 12953 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.
- 12954
- 12955 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.
- 12956
- 12957 Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.
- 12958 Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014).
- 12959 Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1), 1929–1958.
- 12960
- 12961 Srivastava, R. K., K. Greff, and J. Schmidhuber (2015). Training very deep networks. In *Neural Information Processing Systems (NIPS)*, pp. 2377–2385.
- 12962
- 12963 Stab, C. and I. Gurevych (2014a). Annotating argument components and relations in persuasive essays. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp. 1501–1510.
- 12964
- 12965

- 12966 Stab, C. and I. Gurevych (2014b). Identifying argumentative discourse structures in per-
12967 suasive essays. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Lan-*
12968 *guage Processing (EMNLP)*, pp. 46–56.
- 12969 Stede, M. (2011, nov). *Discourse Processing*, Volume 4 of *Synthesis Lectures on Human Lan-*
12970 *guage Technologies*. Morgan & Claypool Publishers.
- 12971 Steedman, M. and J. Baldridge (2011). Combinatory categorial grammar. In *Non-*
12972 *Transformational Syntax: Formal and Explicit Models of Grammar*. Wiley-Blackwell.
- 12973 Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii (2012). Brat: a web-
12974 based tool for nlp-assisted text annotation. In *Proceedings of the European Chapter of the*
12975 *Association for Computational Linguistics (EACL)*, pp. 102–107.
- 12976 Stern, M., J. Andreas, and D. Klein (2017). A minimal span-based neural constituency
12977 parser. In *Proceedings of the Association for Computational Linguistics (ACL)*.
- 12978 Stolcke, A., K. Ries, N. Coccaro, E. Shriberg, R. Bates, D. Jurafsky, P. Taylor, R. Martin,
12979 C. Van Ess-Dykema, and M. Meteer (2000). Dialogue act modeling for automatic tag-
12980 ging and recognition of conversational speech. *Computational linguistics* 26(3), 339–373.
- 12981 Stone, P. J. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. The MIT
12982 Press.
- 12983 Stoyanov, V., N. Gilbert, C. Cardie, and E. Riloff (2009). Conundrums in noun phrase
12984 coreference resolution: Making sense of the state-of-the-art. In *Proceedings of the Associa-*
12985 *tion for Computational Linguistics (ACL)*, pp. 656–664.
- 12986 Strang, G. (2016). *Introduction to linear algebra* (Fifth ed.). Wellesley, MA: Wellesley-
12987 Cambridge Press.
- 12988 Strubell, E., P. Verga, D. Belanger, and A. McCallum (2017). Fast and accurate entity recog-
12989 nition with iterated dilated convolutions. In *Proceedings of Empirical Methods for Natural*
12990 *Language Processing (EMNLP)*.
- 12991 Suchanek, F. M., G. Kasneci, and G. Weikum (2007). Yago: a core of semantic knowledge.
12992 In *Proceedings of the Conference on World-Wide Web (WWW)*, pp. 697–706.
- 12993 Sun, X., T. Matsuzaki, D. Okanohara, and J. Tsujii (2009). Latent variable perceptron algo-
12994 rithm for structured classification. In *Proceedings of the International Joint Conference on*
12995 *Artificial Intelligence (IJCAI)*, Volume 9, pp. 1236–1242.
- 12996 Sun, Y., L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang (2015). Modeling mention, context
12997 and entity with neural networks for entity disambiguation. In *IJCAI*, pp. 1333–1339.

- 12998 Sundermeyer, M., R. Schlüter, and H. Ney (2012). Lstm neural networks for language
 12999 modeling. In *INTERSPEECH*.
- 13000 Surdeanu, M., J. Tibshirani, R. Nallapati, and C. D. Manning (2012). Multi-instance multi-
 13001 label learning for relation extraction. In *Proceedings of Empirical Methods for Natural Lan-*
 13002 *guage Processing (EMNLP)*, pp. 455–465.
- 13003 Sutskever, I., O. Vinyals, and Q. V. Le (2014). Sequence to sequence learning with neural
 13004 networks. In *Neural Information Processing Systems (NIPS)*, pp. 3104–3112.
- 13005 Sutton, R. S. and A. G. Barto (1998). *Reinforcement learning: An introduction*, Volume 1. MIT
 13006 press Cambridge.
- 13007 Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000). Policy gradient methods
 13008 for reinforcement learning with function approximation. In *Neural Information Process-*
 13009 *ing Systems (NIPS)*, pp. 1057–1063.
- 13010 Suzuki, J., S. Takase, H. Kamigaito, M. Morishita, and M. Nagata (2018). An empirical
 13011 study of building a strong baseline for constituency parsing. In *Proceedings of the Asso-*
 13012 *ciation for Computational Linguistics (ACL)*, pp. 612–618.
- 13013 Sweeney, L. (2013). Discrimination in online ad delivery. *Queue* 11(3), 10.
- 13014 Taboada, M., J. Brooke, M. Tofiloski, K. Voll, and M. Stede (2011). Lexicon-based methods
 13015 for sentiment analysis. *Computational linguistics* 37(2), 267–307.
- 13016 Taboada, M. and W. C. Mann (2006). Rhetorical structure theory: Looking back and mov-
 13017 ing ahead. *Discourse studies* 8(3), 423–459.
- 13018 Täckström, O., K. Ganchev, and D. Das (2015). Efficient inference and structured learning
 13019 for semantic role labeling. *Transactions of the Association for Computational Linguistics* 3,
 13020 29–41.
- 13021 Täckström, O., R. McDonald, and J. Uszkoreit (2012). Cross-lingual word clusters for
 13022 direct transfer of linguistic structure. In *Proceedings of the North American Chapter of the*
 13023 *Association for Computational Linguistics (NAACL)*, pp. 477–487.
- 13024 Tang, D., B. Qin, and T. Liu (2015). Document modeling with gated recurrent neural net-
 13025 work for sentiment classification. In *Proceedings of Empirical Methods for Natural Language*
 13026 *Processing (EMNLP)*, pp. 1422–1432.
- 13027 Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin markov networks. In *Neural*
 13028 *Information Processing Systems (NIPS)*.

- 13029 Tausczik, Y. R. and J. W. Pennebaker (2010). The psychological meaning of words: LIWC
13030 and computerized text analysis methods. *Journal of Language and Social Psychology* 29(1),
13031 24–54.
- 13032 Teh, Y. W. (2006). A hierarchical bayesian language model based on pitman-yor processes.
13033 In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 985–992.
- 13034 Tesnière, L. (1966). *Éléments de syntaxe structurale* (second ed.). Paris: Klincksieck.
- 13035 Teufel, S., J. Carletta, and M. Moens (1999). An annotation scheme for discourse-level
13036 argumentation in research articles. In *Proceedings of the European Chapter of the Association*
13037 for Computational Linguistics (EACL), pp. 110–117.
- 13038 Teufel, S. and M. Moens (2002). Summarizing scientific articles: experiments with relevance
13039 and rhetorical status. *Computational linguistics* 28(4), 409–445.
- 13040 Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or opposition
13041 from Congressional floor-debate transcripts. In *Proceedings of Empirical Methods for*
13042 *Natural Language Processing (EMNLP)*, pp. 327–335.
- 13043 Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal*
13044 *Statistical Society. Series B (Methodological)*, 267–288.
- 13045 Titov, I. and J. Henderson (2007). Constituent parsing with incremental sigmoid belief
13046 networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 632–
13047 639.
- 13048 Toutanova, K., D. Klein, C. D. Manning, and Y. Singer (2003). Feature-rich part-of-speech
13049 tagging with a cyclic dependency network. In *Proceedings of the North American Chapter*
13050 of the Association for Computational Linguistics (NAACL).
- 13051 Trivedi, R. and J. Eisenstein (2013). Discourse connectors for latent subjectivity in senti-
13052 ment analysis. In *Proceedings of the North American Chapter of the Association for Compu-*
13053 *tational Linguistics (NAACL)*, pp. 808–813.
- 13054 Tromble, R. W. and J. Eisner (2006). A fast finite-state relaxation method for enforcing
13055 global constraints on sequence decoding. In *Proceedings of the North American Chapter of*
13056 the Association for Computational Linguistics (NAACL), pp. 423.
- 13057 Tschantaridis, I., T. Hofmann, T. Joachims, and Y. Altun (2004). Support vector machine
13058 learning for interdependent and structured output spaces. In *Proceedings of the twenty-*
13059 *first international conference on Machine learning*, pp. 104. ACM.
- 13060 Tsvetkov, Y., M. Faruqui, W. Ling, G. Lample, and C. Dyer (2015). Evaluation of word
13061 vector representations by subspace alignment. In *Proceedings of Empirical Methods for*
13062 *Natural Language Processing (EMNLP)*, pp. 2049–2054.

- 13063 Tu, Z., Z. Lu, Y. Liu, X. Liu, and H. Li (2016). Modeling coverage for neural machine
13064 translation. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 76–
13065 85.
- 13066 Turian, J., L. Ratinov, and Y. Bengio (2010). Word representations: a simple and general
13067 method for semi-supervised learning. In *Proceedings of the Association for Computational
13068 Linguistics (ACL)*, pp. 384–394.
- 13069 Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*,
13070 pp. 23–65. Springer.
- 13071 Turney, P. D. and P. Pantel (2010). From frequency to meaning: Vector space models of
13072 semantics. *Journal of Artificial Intelligence Research* 37, 141–188.
- 13073 Tutin, A. and R. Kittredge (1992). Lexical choice in context: generating procedural texts.
13074 In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pp.
13075 763–769.
- 13076 Twain, M. (1997). *A Tramp Abroad*. New York: Penguin.
- 13077 Tzeng, E., J. Hoffman, T. Darrell, and K. Saenko (2015). Simultaneous deep transfer across
13078 domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*,
13079 pp. 4068–4076.
- 13080 Usunier, N., D. Buffoni, and P. Gallinari (2009). Ranking with ordered weighted pairwise
13081 classification. In *Proceedings of the International Conference on Machine Learning (ICML)*,
13082 pp. 1057–1064.
- 13083 Uthus, D. C. and D. W. Aha (2013). The ubuntu chat corpus for multiparicipant chat
13084 analysis. In *AAAI Spring Symposium: Analyzing Microtext*, Volume 13, pp. 01.
- 13085 Utiyama, M. and H. Isahara (2001). A statistical model for domain-independent text seg-
13086 mentation. In *Proceedings of the 39th Annual Meeting on Association for Computational
13087 Linguistics*, pp. 499–506. Association for Computational Linguistics.
- 13088 Utiyama, M. and H. Isahara (2007). A comparison of pivot methods for phrase-based
13089 statistical machine translation. In *Human Language Technologies 2007: The Conference of
13090 the North American Chapter of the Association for Computational Linguistics; Proceedings of
13091 the Main Conference*, pp. 484–491.
- 13092 Uzuner, Ö., X. Zhang, and T. Sibanda (2009). Machine learning and rule-based approaches
13093 to assertion classification. *Journal of the American Medical Informatics Association* 16(1),
13094 109–115.

- 13095 Vadas, D. and J. R. Curran (2011). Parsing noun phrases in the penn treebank. *Computational Linguistics* 37(4), 753–809.
- 13096
- 13097 Van Eynde, F. (2006). NP-internal agreement and the structure of the noun phrase. *Journal of Linguistics* 42(1), 139–186.
- 13098
- 13099 Van Gael, J., A. Vlachos, and Z. Ghahramani (2009). The infinite hmm for unsupervised pos tagging. In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 678–687.
- 13100
- 13101
- 13102 Vaswani, A., S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit (2018). Tensor2tensor for neural machine translation. *CoRR abs/1803.07416*.
- 13103
- 13104
- 13105 Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). Attention is all you need. In *Neural Information Processing Systems (NIPS)*, pp. 6000–6010.
- 13106
- 13107
- 13108 Velldal, E., L. Øvrelid, J. Read, and S. Oepen (2012). Speculation and negation: Rules, rankers, and the role of syntax. *Computational linguistics* 38(2), 369–410.
- 13109
- 13110 Versley, Y. (2011). Towards finer-grained tagging of discourse connectives. In *Proceedings of the Workshop Beyond Semantics: Corpus-based Investigations of Pragmatic and Discourse Phenomena*, pp. 2–63.
- 13111
- 13112
- 13113 Vilain, M., J. Burger, J. Aberdeen, D. Connolly, and L. Hirschman (1995). A model-theoretic coreference scoring scheme. In *Proceedings of the 6th conference on Message understanding*, pp. 45–52. Association for Computational Linguistics.
- 13114
- 13115
- 13116 Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11(Dec), 3371–3408.
- 13117
- 13118
- 13119 Vincze, V., G. Szarvas, R. Farkas, G. Móra, and J. Csirik (2008). The bioscope corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC bioinformatics* 9(11), S9.
- 13120
- 13121
- 13122 Vinyals, O., A. Toshev, S. Bengio, and D. Erhan (2015). Show and tell: A neural image caption generator. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pp. 3156–3164. IEEE.
- 13123
- 13124
- 13125 Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13(2), 260–269.
- 13126

- 13127 Voll, K. and M. Taboada (2007). Not all words are created equal: Extracting semantic
 13128 orientation as a function of adjective relevance. In *Proceedings of Australian Conference
 13129 on Artificial Intelligence*.
- 13130 Wager, S., S. Wang, and P. S. Liang (2013). Dropout training as adaptive regularization. In
 13131 *Neural Information Processing Systems (NIPS)*, pp. 351–359.
- 13132 Wainwright, M. J. and M. I. Jordan (2008). Graphical models, exponential families, and
 13133 variational inference. *Foundations and Trends® in Machine Learning* 1(1-2), 1–305.
- 13134 Walker, M. A. (2000). An application of reinforcement learning to dialogue strategy selec-
 13135 tion in a spoken dialogue system for email. *Journal of Artificial Intelligence Research* 12,
 13136 387–416.
- 13137 Walker, M. A., J. E. Cahn, and S. J. Whittaker (1997). Improvising linguistic style: Social
 13138 and affective bases for agent personality. In *Proceedings of the first international conference
 13139 on Autonomous agents*, pp. 96–105. ACM.
- 13140 Wang, C., N. Xue, and S. Pradhan (2015). A Transition-based Algorithm for AMR Parsing.
 13141 In *Proceedings of the North American Chapter of the Association for Computational Linguistics
 13142 (NAACL)*, pp. 366–375.
- 13143 Wang, H., T. Onishi, K. Gimpel, and D. McAllester (2017). Emergent predication structure
 13144 in hidden state vectors of neural readers. In *Proceedings of the 2nd Workshop on Represen-
 13145 tation Learning for NLP*, pp. 26–36.
- 13146 Weaver, W. (1955). Translation. *Machine translation of languages* 14, 15–23.
- 13147 Webber, B. (2004, sep). D-LTAG: extending lexicalized TAG to discourse. *Cognitive Sci-
 13148 ence* 28(5), 751–779.
- 13149 Webber, B., M. Egg, and V. Kordoni (2012). Discourse structure and language technology.
 13150 *Journal of Natural Language Engineering* 1.
- 13151 Webber, B. and A. Joshi (2012). Discourse structure and computation: past, present and
 13152 future. In *Proceedings of the ACL-2012 Special Workshop on Rediscovering 50 Years of Dis-
 13153 coveries*, pp. 42–54. Association for Computational Linguistics.
- 13154 Wei, G. C. and M. A. Tanner (1990). A monte carlo implementation of the em algorithm
 13155 and the poor man’s data augmentation algorithms. *Journal of the American Statistical
 13156 Association* 85(411), 699–704.
- 13157 Weinberger, K., A. Dasgupta, J. Langford, A. Smola, and J. Attenberg (2009). Feature
 13158 hashing for large scale multitask learning. In *Proceedings of the International Conference
 13159 on Machine Learning (ICML)*, pp. 1113–1120.

- 13160 Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language
13161 communication between man and machine. *Communications of the ACM* 9(1), 36–45.
- 13162 Wellner, B. and J. Pustejovsky (2007). Automatically identifying the arguments of dis-
13163 course connectives. In *Proceedings of Empirical Methods for Natural Language Processing*
13164 (*EMNLP*), pp. 92–101.
- 13165 Wen, T.-H., M. Gasic, N. Mrkšić, P.-H. Su, D. Vandyke, and S. Young (2015). Semantically
13166 conditioned lstm-based natural language generation for spoken dialogue systems. In
13167 *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 1711–1721.
- 13168 Weston, J., S. Bengio, and N. Usunier (2011). Wsabie: Scaling up to large vocabulary image
13169 annotation. In *IJCAI*, Volume 11, pp. 2764–2770.
- 13170 Wiebe, J., T. Wilson, and C. Cardie (2005). Annotating expressions of opinions and emo-
13171 tions in language. *Language resources and evaluation* 39(2), 165–210.
- 13172 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2015). Towards universal paraphrastic
13173 sentence embeddings. *arXiv preprint arXiv:1511.08198*.
- 13174 Wieting, J., M. Bansal, K. Gimpel, and K. Livescu (2016). CHARAGRAM: Embedding
13175 words and sentences via character n-grams. In *Proceedings of Empirical Methods for Nat-*
13176 *ural Language Processing (EMNLP)*, pp. 1504–1515.
- 13177 Williams, J. D. and S. Young (2007). Partially observable markov decision processes for
13178 spoken dialog systems. *Computer Speech & Language* 21(2), 393–422.
- 13179 Williams, P., R. Sennrich, M. Post, and P. Koehn (2016). Syntax-based statistical machine
13180 translation. *Synthesis Lectures on Human Language Technologies* 9(4), 1–208.
- 13181 Wilson, T., J. Wiebe, and P. Hoffmann (2005). Recognizing contextual polarity in phrase-
13182 level sentiment analysis. In *Proceedings of Empirical Methods for Natural Language Pro-*
13183 *cessing (EMNLP)*, pp. 347–354.
- 13184 Winograd, T. (1972). Understanding natural language. *Cognitive psychology* 3(1), 1–191.
- 13185 Wiseman, S., A. M. Rush, and S. M. Shieber (2016). Learning global features for corefer-
13186 ence resolution. In *Proceedings of the North American Chapter of the Association for Compu-*
13187 *tational Linguistics (NAACL)*, pp. 994–1004.
- 13188 Wiseman, S., S. Shieber, and A. Rush (2017). Challenges in data-to-document generation.
13189 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*, pp. 2253–
13190 2263.

- 13191 Wiseman, S. J., A. M. Rush, S. M. Shieber, and J. Weston (2015). Learning anaphoricity and
 13192 antecedent ranking features for coreference resolution. In *Proceedings of the Association
 13193 for Computational Linguistics (ACL)*.
- 13194 Wolf, F. and E. Gibson (2005). Representing discourse coherence: A corpus-based study.
 13195 *Computational Linguistics* 31(2), 249–287.
- 13196 Wolfe, T., M. Dredze, and B. Van Durme (2017). Pocket knowledge base population. In
 13197 *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 305–310.
- 13198 Wong, Y. W. and R. Mooney (2007). Generation by inverting a semantic parser that uses
 13199 statistical machine translation. In *Proceedings of the North American Chapter of the Associa-
 13200 tion for Computational Linguistics (NAACL)*, pp. 172–179.
- 13201 Wong, Y. W. and R. J. Mooney (2006). Learning for semantic parsing with statistical ma-
 13202 chine translation. In *Proceedings of the North American Chapter of the Association for Com-
 13203 putational Linguistics (NAACL)*, pp. 439–446.
- 13204 Wu, B. Y. and K.-M. Chao (2004). *Spanning trees and optimization problems*. CRC Press.
- 13205 Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of par-
 13206 allel corpora. *Computational linguistics* 23(3), 377–403.
- 13207 Wu, F. and D. S. Weld (2010). Open information extraction using wikipedia. In *Proceedings
 13208 of the Association for Computational Linguistics (ACL)*, pp. 118–127.
- 13209 Wu, X., R. Ward, and L. Bottou (2018). Wngrad: Learn the learning rate in gradient de-
 13210 scent. *arXiv preprint arXiv:1803.02865*.
- 13211 Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao,
 13212 Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws,
 13213 Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young,
 13214 J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean (2016).
 13215 Google’s neural machine translation system: Bridging the gap between human and ma-
 13216 chine translation. *CoRR abs/1609.08144*.
- 13217 Xia, F. (2000). The part-of-speech tagging guidelines for the penn chinese treebank (3.0).
 13218 Technical report, University of Pennsylvania Institute for Research in Cognitive Science.
- 13219 Xu, K., J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio
 13220 (2015). Show, attend and tell: Neural image caption generation with visual attention.
 13221 In *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 2048–2057.
- 13222 Xu, W., X. Liu, and Y. Gong (2003). Document clustering based on non-negative matrix
 13223 factorization. In *SIGIR*, pp. 267–273. ACM.

- 13224 Xu, Y., L. Mou, G. Li, Y. Chen, H. Peng, and Z. Jin (2015). Classifying relations via long
13225 short term memory networks along shortest dependency paths. In *Proceedings of Empirical
13226 Methods for Natural Language Processing (EMNLP)*, pp. 1785–1794.
- 13227 Xuan Bach, N., N. L. Minh, and A. Shimazu (2012). A reranking model for discourse seg-
13228 mentation using subtree features. In *Proceedings of the Special Interest Group on Discourse
13229 and Dialogue (SIGDIAL)*.
- 13230 Xue, N. et al. (2003). Chinese word segmentation as character tagging. *Computational
13231 Linguistics and Chinese Language Processing* 8(1), 29–48.
- 13232 Xue, N., H. T. Ng, S. Pradhan, R. Prasad, C. Bryant, and A. T. Rutherford (2015). The
13233 CoNLL-2015 shared task on shallow discourse parsing. In *Proceedings of the Conference
13234 on Natural Language Learning (CoNLL)*.
- 13235 Yamada, H. and Y. Matsumoto (2003). Statistical dependency analysis with support vector
13236 machines. In *Proceedings of IWPT*, Volume 3, pp. 195–206.
- 13237 Yamada, K. and K. Knight (2001). A syntax-based statistical translation model. In *Proceed-
13238 ings of the 39th Annual Meeting on Association for Computational Linguistics*, pp. 523–530.
13239 Association for Computational Linguistics.
- 13240 Yang, B. and C. Cardie (2014). Context-aware learning for sentence-level sentiment anal-
13241 ysis with posterior regularization. In *Proceedings of the Association for Computational Lin-
13242 guistics (ACL)*.
- 13243 Yang, Y., M.-W. Chang, and J. Eisenstein (2016). Toward socially-infused information ex-
13244 traction: Embedding authors, mentions, and entities. In *Proceedings of Empirical Methods
13245 for Natural Language Processing (EMNLP)*.
- 13246 Yang, Y. and J. Eisenstein (2013). A log-linear model for unsupervised text normalization.
13247 In *Proceedings of Empirical Methods for Natural Language Processing (EMNLP)*.
- 13248 Yang, Y. and J. Eisenstein (2015). Unsupervised multi-domain adaptation with feature em-
13249 beddings. In *Proceedings of the North American Chapter of the Association for Computational
13250 Linguistics (NAACL)*.
- 13251 Yang, Y., W.-t. Yih, and C. Meek (2015). WikiQA: A challenge dataset for open-domain
13252 question answering. In *Proceedings of Empirical Methods for Natural Language Processing
13253 (EMNLP)*, pp. 2013–2018.
- 13254 Yannakoudakis, H., T. Briscoe, and B. Medlock (2011). A new dataset and method for
13255 automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Associa-
13256 tion for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 180–189.
13257 Association for Computational Linguistics.

- 13258 Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised meth-
 13259 ods. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 189–196.
 13260 Association for Computational Linguistics.
- 13261 Yee, L. C. and T. Y. Jones (2012, March). Apple ceo in china mission to clear up problems.
 13262 *Reuters*. retrieved on March 26, 2017.
- 13263 Yi, Y., C.-Y. Lai, S. Petrov, and K. Keutzer (2011, October). Efficient parallel cky parsing on
 13264 gpus. In *Proceedings of the 12th International Conference on Parsing Technologies*, Dublin,
 13265 Ireland, pp. 175–185. Association for Computational Linguistics.
- 13266 Yu, C.-N. J. and T. Joachims (2009). Learning structural svms with latent variables. In
 13267 *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 1169–1176.
- 13268 Yu, F. and V. Koltun (2016). Multi-scale context aggregation by dilated convolutions. In
 13269 *Proceedings of the International Conference on Learning Representations (ICLR)*.
- 13270 Zaidan, O. F. and C. Callison-Burch (2011). Crowdsourcing translation: Professional qual-
 13271 ity from non-professionals. In *Proceedings of the Association for Computational Linguistics*
 13272 (*ACL*), pp. 1220–1229.
- 13273 Zaremba, W., I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv*
 13274 preprint *arXiv:1409.2329*.
- 13275 Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint*
 13276 *arXiv:1212.5701*.
- 13277 Zelenko, D., C. Aone, and A. Richardella (2003). Kernel methods for relation extraction.
 13278 *The Journal of Machine Learning Research* 3, 1083–1106.
- 13279 Zelle, J. M. and R. J. Mooney (1996). Learning to parse database queries using induc-
 13280 tive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*
 13281 (*AAAI*), pp. 1050–1055.
- 13282 Zeng, D., K. Liu, S. Lai, G. Zhou, and J. Zhao (2014). Relation classification via convolu-
 13283 tional deep neural network. In *Proceedings of the International Conference on Computational*
 13284 *Linguistics (COLING)*, pp. 2335–2344.
- 13285 Zettlemoyer, L. S. and M. Collins (2005). Learning to map sentences to logical form: Struc-
 13286 tured classification with probabilistic categorial grammars. In *Proceedings of UAI*.
- 13287 Zhang, X., J. Zhao, and Y. LeCun (2015). Character-level convolutional networks for text
 13288 classification. In *Neural Information Processing Systems (NIPS)*, pp. 649–657.

- 13289 Zhang, Y. and S. Clark (2008). A tale of two parsers: investigating and combining graph-
13290 based and transition-based dependency parsing using beam-search. In *Proceedings of*
13291 *Empirical Methods for Natural Language Processing (EMNLP)*, pp. 562–571.
- 13292 Zhang, Y., T. Lei, R. Barzilay, T. Jaakkola, and A. Globerson (2014). Steps to excellence:
13293 Simple inference with refined scoring of dependency trees. In *Proceedings of the Associa-*
13294 *tion for Computational Linguistics (ACL)*, pp. 197–207.
- 13295 Zhang, Y. and J. Nivre (2011). Transition-based dependency parsing with rich non-local
13296 features. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 188–193.
- 13297 Zhang, Z. (2017). A note on counting dependency trees. *arXiv preprint arXiv:1708.08789*.
- 13298 Zhao, J., T. Wang, M. Yatskar, e. V. Ordonez, and K.-W. Chang (2018). Gender bias in
13299 coreference resolution: Evaluation and debiasing methods. In *Proceedings of the North*
13300 *American Chapter of the Association for Computational Linguistics (NAACL)*.
- 13301 Zhou, J. and W. Xu (2015). End-to-end learning of semantic role labeling using recurrent
13302 neural networks. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp.
13303 1127–1137.
- 13304 Zhu, J., Z. Nie, X. Liu, B. Zhang, and J.-R. Wen (2009). Statsnowball: a statistical approach
13305 to extracting entity relationships. In *Proceedings of the Conference on World-Wide Web*
13306 (*WWW*), pp. 101–110.
- 13307 Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). Semi-supervised learning using gaus-
13308 sian fields and harmonic functions. In *Proceedings of the International Conference on Ma-*
13309 *chine Learning (ICML)*, pp. 912–919.
- 13310 Zhu, X. and A. B. Goldberg (2009). Introduction to semi-supervised learning. *Synthesis*
13311 *lectures on artificial intelligence and machine learning* 3(1), 1–130.
- 13312 Zipf, G. K. (1949). Human behavior and the principle of least effort.
- 13313 Zirn, C., M. Niepert, H. Stuckenschmidt, and M. Strube (2011). Fine-grained sentiment
13314 analysis with structural features. In *IJCNLP*, Chiang Mai, Thailand, pp. 336–344.
- 13315 Zou, W. Y., R. Socher, D. Cer, and C. D. Manning (2013). Bilingual word embeddings
13316 for phrase-based machine translation. In *Proceedings of Empirical Methods for Natural*
13317 *Language Processing (EMNLP)*, pp. 1393–1398.

¹³³¹⁸ Index

- 13319 α -conversion, 295
13320 β -reduction, β -conversion, 293
13321 *n*-gram, 24
13322 language model, 198
13323 *p*-value, 84
13324 one-tailed, 85
13325 *F*-MEASURE, 82
13326 balanced, 83
13327 macro, 83
13328 micro, 83
13329 WORDNET, 9, 74
13330 BLEU, 433
13331 METEOR, 435
13332 RIBES, 435
13333 semantics
13334 extra-propositional, 422
13335 ablation test, 84
13336 Abstract Meaning Representation (AMR), 307, 319
13337 accuracy, 23, 81
13339 action, in reinforcement learning, 365
13340 AdaGrad, 39, 61
13341 adequacy, in translation, 433
13342 adjectives, 177
13343 adjuncts, 306
13344 adpositions, 178
13345 adverbs, 177
13346 affix, 195
13347 inflectional, 78
13348 agent (thematic role), 308
13349 alignment
13350 in machine translation, 432, 437
13351 in semantic parsing, 321
13352 in text generation, 459
13353 Amazon Mechanical Turk, 91
13354 ambiguity, 210, 218
13355 attachment, 257
13356 derivational, 222
13357 spurious, 222, 269, 298
13358 syntactic, 229
13359 anaphoricity, 360
13360 anchored productions, 232
13361 animacy, 307
13362 annealing, 452
13363 antecedent mention, 351, 359
13364 antonymy, 74, 281
13365 apophony, 194
13366 area under the curve (AUC), 83
13367 argumentation, 392
13368 arguments, 403
13369 article, 182
13370 aspect, 177
13371 aspect-based opinion mining, 71
13372 autoencoder, 345
13373 denoising, 345, 408
13374 variational, 465
13375 automated theorem provers, 288
13376 automatic differentiation, 56
13377 auxiliary verbs, 178
13378 average mutual information, 333
13379 averaged perceptron, 27

- 13380 backchannel, 187
 13381 backoff, 130
 Katz, 130
 13383 backpropagation, 55
 through time, 136
 13385 backward recurrence, 165, 166
 13386 backward-looking center, 382
 13387 bag of words, 13
 13388 balanced test set, 81
 13389 batch normalization, 60
 13390 Baum-Welch algorithm, 170
 13391 Bayes' rule, 478
 13392 Bayesian nonparametrics, 103, 249
 13393 beam sampling, 172
 13394 beam search, 272
 in coreference resolution, 364
 in machine translation, 449, 450
 13395 Bell number, 362
 13398 bias, 22
 13399 bias-variance tradeoff, 22, 127, 129
 13400 bigrams, 24, 70
 13401 bilinear product, 330
 13402 binarization, 211, 228
 13403 binomial
 distribution, 85
 random variable, 482
 13406 test, 85
 13407 BIO notation, 184, 317
 13408 biomedical natural language processing,
 183
 13410 Bonferroni correction, 87
 13411 boolean semiring, 199
 13412 boosting, 48
 13413 bootstrap samples, 86
 13414 brevity penalty, 434
 13415 Brown clusters, 327, 332
 13416 byte-pair encoding, 343, 448
 13417 c-command, 353
 13418 case marking, 182, 219
 13419 Catalan number, 225
 13420 cataphora, 352
 13421 center embedding, 207
 13422 centering theory, 355, 382
 13423 character-level language models, 141
 13424 chatbots, 470
 13425 Chomsky Normal Form (CNF), 211
 13426 Chu-Liu-Edmonds algorithm, 264
 13427 CKY algorithm, 226
 13428 class imbalance, 81
 13429 classification, 13
 large margin, 30
 lexicon-based, 72
 weights, 14
 13433 closed-vocabulary, 141
 13434 closure, of classes of formal languages,
 192
 13435 cluster ranking, 363
 13437 clustering, 96
 K-means, 96
 exchange, 333
 hierarchical, 332
 soft, 97
 13442 co-hypernymy, 281
 13443 co-training, 108
 13444 code switching, 180, 186
 13445 Cohen's Kappa, 90
 13446 coherence, 396
 13447 cohesion, 379, 400
 13448 collective entity linking, 408
 13449 collocation extraction, 344
 13450 combinatory categorial grammar, 220
 13451 complement clause, 213
 13452 complement event (probability), 476
 13453 composition (CCG), 221
 13454 compositional vector grammars, 391
 13455 compositionality, 2, 7, 10, 291, 341
 13456 computation graph, 48, 55
 dynamic, 56
 13458 computational linguistics (versus
 natural language processing), 1
 13460 computational social science, 5

- 13461 concept (in Abstract Meaning Representation), 319
 13463 conditional independence, 154, 480
 13464 conditional random field, 162
 13465 confidence interval, 86
 13466 configuration (transition-based parsing), 269
 13468 consistency, in logic, 291
 13469 constants, in logic, 286
 13470 constituents, 212
 13471 split, 312
 13472 constrained optimization, 315
 13473 content selection, 457
 13474 content words, 178
 13475 context-free grammars, 208
 13476 probabilistic (PCFGs), 235
 13477 synchronous, 441
 13478 weighted, 218, 227, 233
 13479 context-free languages, 207, 208
 13480 context-free production, 209
 13481 recursive, 209
 13482 unary, 210
 13483 context-sensitive languages, 218
 13484 continuous bag-of-words (CBOW), 334
 13485 contradiction, 346
 13486 conversational turns, 187
 13487 convexity, 29, 58, 300, 482, 485
 13488 biconvexity, 102
 13489 convolution
 13490 dilated, 63, 185
 13491 narrow, 63
 13492 one-dimensional, 63
 13493 wide, 63
 13494 convolutional neural networks, 61
 13495 in machine translation, 447
 13496 in relation extraction, 415
 13497 in semantic role labeling, 318
 13498 in sequence labeling, 170
 13499 cooperative principle, 351
 13500 coordinate ascent, 102
 13501 coordinating conjunctions, 178
 13502 copula, 177, 217, 260
 13503 coreference resolution, 347, 351
 13504 cross-document, 406
 13505 coreferent, 351
 13506 cosine similarity, 339, 380
 13507 cost-augmented decoding, 33, 34
 13508 coverage loss, 465
 13509 critical point, 58, 485
 13510 cross-entropy, 53, 416
 13511 cross-serial dependencies, 219
 13512 cross-validation, 23
 13513 crowdsourcing, 91
 13514 cumulative probability distribution, 85
 13515 decidability, 291
 13516 decision trees, 48
 13517 decoding
 13518 cost-augmented, 161
 in conditional random fields, 163
 13519 definiteness, 183
 13520 delta function, 22
 13521 denotation, 286
 13523 dependency, 258
 grammar, 257
 graph, 258
 labels, 259
 path, 75, 314, 413
 syntactic, 258
 13524 dependency parsing, 257
 13525 arc-eager, 269, 270
 13526 arc-factored, 263
 13527 arc-standard, 269
 13528 pseudo-projective, 272
 13529 second-order, 263
 13530 third-order, 264
 13531 derivation
 in context-free languages, 209
 in dependency parsing, 269
 in semantic parsing, 293
 13532 determiner, 179
 13533 phrase, 215
 13534

- 13542 development set, 23, 81
 13543 dialogue acts, 90, 187, 471
 13544 dialogue management, 468
 13545 dialogue systems, 125, 466
 agenda-based, 468
 mixed-initiative, 468
 13547 digital humanities, 5, 69
 13549 Dirichlet Compound Multinomial, 398
 13550 Dirichlet distribution, 115
 13551 discounting, 130
 absolute, 130
 13553 discourse, 379
 connectives, 385
 13555 depth, 394
 13556 parsing, 385
 13557 segment, 379
 13558 unit, 389
 13559 discourse relations, 347, 384
 13560 coordinating, 389
 13561 implicit, 385
 13562 sense classification, 386
 13563 subordinating, 389
 13564 distant supervision, 118, 418, 419
 13565 distributed semantics, 327
 13566 distributional
 hypothesis, 325, 326
 semantics, 10, 327
 statistics, 75, 248, 326
 13569 document frequency, 407
 13571 domain adaptation, 95, 111
 by projection, 112
 13573 dropout, 57, 137
 13574 dual decomposition, 317
 13575 dynamic programming, 149
- 13576 E-step, 99
 13577 early stopping, 28, 61
 13578 early update, 276
 13579 edit distance, 201, 435
 13580 effective counts, 129
 13581 elementwise nonlinearity, 50
- 13582 Elman unit, 135
 13583 ELMo (embeddings from language models), 340
 13584 embedding, 167
 13586 emotion, 72
 13587 empirical Bayes, 116
 13588 empty string, 192
 13589 encoder-decoder model, 345, 442, 460
 13590 ensemble learning, 48, 318, 444
 13591 entailment, 290, 346
 13592 entity, 403
 embeddings, 407
 13594 grid, 383
 13595 linking, 351, 403, 405, 416
 13596 linking, collective, 409
 13597 nil, 405
 13598 entropy, 41, 99
 13599 estimation, 482
 13600 EuroParl corpus, 435
 13601 evaluation
 extrinsic, 139, 338
 intrinsic, 139, 338
 13603 event, 403
 13605 coreference, 421
 13606 detection, 420
 13607 event semantics, 305
 13608 events, in probability, 475
 13609 disjoint, 476
 13610 evidentiality, 182, 423
 13611 expectation, 481
 13612 expectation-maximization, 97
 13613 hard, 102
 13614 in language modeling, 131
 13615 in machine translation, 439
 13616 incremental, 102
 13617 online, 102
 13618 explicit semantic analysis, 328
- 13619 factoid question, 424
 13620 factor graph, 163
 13621 factuality, 423

- 13622 fairness and bias, 5
 13623 in machine translation, 434
 13624 in word embeddings, 340
 13625 false discovery rate, 87
 13626 false negative, 81
 rate, 478
 13627 false positive, 81, 479
 rate, 83, 478
 13629 feature
 13631 co-adaptation, 58
 13632 function, 14
 13633 hashing, 80
 13634 noising, 58
 13635 selection, 40
 13636 features, 6
 13637 bilexical, 266
 13638 collocation, 75
 13639 emission, 148
 13640 lexical, 47
 13641 offset, 15
 13642 pivot, 113
 13643 transition, 148
 13644 finite state
 13645 acceptor, 193
 13646 acceptor, chain, 205
 13647 acceptors, weighted, 197
 13648 automata, 193
 13649 automaton, deterministic, 194
 13650 composition, 204
 13651 transducers, 196, 201
 13652 finite-state
 13653 transduction, 192
 13654 fluency, 125, 433
 13655 formal language theory, 191
 13656 forward
 13657 recurrence, 164
 13658 variable, 164, 166
 13659 forward-backward algorithm, 165, 206,
 240
 13660 forward-looking centers, 382
 13661 frame, 310
 13663 element, 310
 13664 in dialogue systems, 467
 13665 FrameNet, 310
 13666 Frobenius norm, 57, 105
 13667 function words, 178
 13668 function, in logic, 289
 13669 functional segmentation, 379, 381
 13670 garden path sentence, 146
 13671 gazetteer, 184, 357, 413
 13672 generalization, 28
 13673 generalized linear models, 41
 13674 generative model
 for classification, 17
 for coreference, 363
 for interpolated language modeling,
 131
 for parsing, 235
 for sequence labeling, 154
 13675 Gibbs sampling, 115, 409
 13676 collapsed, 116
 13677 gloss, 125, 179, 433
 13678 government and binding theory, 353
 13679 gradient, 29
 13680 clipping, 60
 13681 descent, 37
 13682 exploding, 137
 13683 vanishing, 51, 137
 13684 Gram matrix, 414
 13685 grammar equivalence, 210
 13686 grammar induction, 241
 13687 grammaticality, 396
 13688 graphical model, 154
 13689 graphics processing units (GPUs), 170,
 185
 13690 grid search, 23
 13691 Hamming cost, 161
 13692 Hansards corpus, 435
 13693 hanzi, 77
 13694 head rules, for lexicalized CFGs, 246, 257

- 13702 head word, 212, 244, 257, 353
- 13703 of a dependency edge, 258
- 13704 hedging, 423
- 13705 held-out data, 139
- 13706 Hessian matrix, 38
- 13707 hidden Markov models, 154
- 13708 hierarchical recurrent network, 470
- 13709 highway network, 52
- 13710 holonymy, 75
- 13711 homonymy, 73
- 13712 human computation, 91
- 13713 hypergraph, 393
- 13714 hypernymy, 75, 281
- 13715 hyperparameter, 22
- 13716 hyponymy, 75
- 13717 illocutionary force, 187
- 13718 importance sampling, 452
- 13719 importance score, 452
- 13720 independent and identically distributed (IID), 16
- 13721 inference
- 13723 in structured prediction, 147
- 13724 logical, 285
- 13725 rules for propositional logic, 288
- 13726 inflection point, 486
- 13727 information extraction, 403
- 13728 open, 419
- 13729 information retrieval, 5, 416
- 13730 inside recurrence, 236, 237
- 13731 inside-outside algorithm, 240, 249
- 13732 instance labels, 16
- 13733 instance, in Abstract Meaning Representation, 319
- 13735 integer linear programming, 315
- 13736 in coreference resolution, 362
- 13737 in entity linking, 409
- 13738 in extractive summarization, 395
- 13739 in sentence compression, 466
- 13740 inter-annotator agreement, 90
- 13741 interjections, 177
- 13742 interlingua, 432
- 13743 interpolation, 131, 199
- 13744 interval algebra, 421
- 13745 inverse document frequency, 407
- 13746 inversion (of finite state automata), 203
- 13747 irrealis, 70
- 13748 Jensen's inequality, 99
- 13749 Kalman smoother, 172
- 13750 kernel, 48, 414
- 13751 Kleene star, 192
- 13752 knapsack problem, 395
- 13753 knowledge base, 403
- 13754 population, 416
- 13755 label bias problem, 275, 280
- 13756 label propagation, 110, 120
- 13757 Lagrangian, 487
- 13758 lambda calculus, 292
- 13759 lambda expressions, 292
- 13760 language model, 4, 126
- 13761 latent Dirichlet allocation, 380
- 13762 latent semantic analysis, 328, 329
- 13763 latent variable, 98, 206, 299, 419, 432
- 13764 conditional random field, 301
- 13765 in parsing, 249
- 13766 perceptron, 207, 300, 360
- 13767 layer normalization, 61, 447
- 13768 learning
- 13769 active, 118
- 13770 batch, 25
- 13771 constraint-driven, 118
- 13772 deep, 47
- 13773 discriminative, 25
- 13774 multiple instance, 118, 418
- 13775 multitask, 118
- 13776 online, 25, 38
- 13777 reinforcement, 365, 450
- 13778 semi-supervised, 76, 95, 339
- 13779 to search, 257, 277, 366
- 13780 transfer, 118

- | | | | |
|-------|--|-------|-------------------------------------|
| 13781 | unsupervised, 71, 95 | 13822 | loss |
| 13782 | learning rate, 38 | 13823 | function, 28 |
| 13783 | least squares, 72 | 13824 | hinge, 29 |
| 13784 | leave-one-out cross-validation, 24 | 13825 | logistic, 35 |
| 13785 | lemma, 73, 202 | 13826 | WARP, 411 |
| 13786 | lemmatization, 79 | 13827 | zero-one, 28 |
| 13787 | lexical entry, 293 | 13828 | machine learning, 2 |
| 13788 | lexical unit, 310 | 13829 | supervised, 16 |
| 13789 | lexicalization | 13830 | theory, 26 |
| 13790 | in parsing, 245 | 13831 | machine reading, 425 |
| 13791 | in text generation, 457 | 13832 | macro, 404 |
| 13792 | lexicalized tree-adjoining grammar for | 13833 | micro, 404 |
| 13793 | discourse (D-LTAG), 385 | 13834 | machine translation, 125 |
| 13794 | lexicon, 293 | 13835 | neural, 432 |
| 13795 | in combinatory categorial grammar, | 13836 | statistical, 432 |
| 13796 | 221 | 13837 | margin, 25, 30 |
| 13797 | lexicon-based classification, 72 | 13838 | functional, 31 |
| 13798 | seed, 73 | 13839 | geometric, 31 |
| 13799 | light verb, 321 | 13840 | marginal relevance, 465 |
| 13800 | linear separability, 25 | 13841 | marginalization, 477 |
| 13801 | linearization, 466, 472 | 13842 | markable, 358 |
| 13802 | link function, 41 | 13843 | Markov assumption, 154 |
| 13803 | literal character, 192 | 13844 | Markov blanket, 154 |
| 13804 | local optimum, 102, 486 | 13845 | Markov Chain Monte Carlo (MCMC), |
| 13805 | locally-normalized objective, 275 | 13846 | 103, 115, 172 |
| 13806 | log-bilinear language model, 342 | 13847 | Markov decision process, 468 |
| 13807 | log-likelihood | 13848 | partially-observable (POMDP), 470 |
| 13808 | conditional, 52 | 13849 | Markov random fields, 162 |
| 13809 | log-linear models, 41 | 13850 | Markovization |
| 13810 | logic, 287 | 13851 | vertical, 244 |
| 13811 | first-order, 288 | 13852 | matrix-tree theorem, 268 |
| 13812 | higher-order, 289 | 13853 | max-margin Markov network, 161 |
| 13813 | propositional, 287 | 13854 | max-product algorithm, 157 |
| 13814 | logistic function, 41 | 13855 | maximum a posteriori, 22, 483 |
| 13815 | long short-term memory (LSTM), 52, | 13856 | maximum conditional likelihood, 35 |
| 13816 | 135, 137, 181, 442 | 13857 | maximum directed spanning tree, 264 |
| 13817 | bidirectional, 169, 445 | 13858 | maximum entropy, 41 |
| 13818 | deep, 443 | 13859 | maximum likelihood, 17, 21, 482 |
| 13819 | LSTM-CRF, 169, 318 | 13860 | McNemar's test, 85 |
| 13820 | memory cell, 137 | 13861 | meaning representation, 285 |
| 13821 | lookup layer, 53, 135 | | |

- | | | | |
|-------|---|-------|---|
| 13862 | membership problem, 191 | 13902 | linking, 405 |
| 13863 | mention | 13903 | recognition, 169, 403, 405 |
| 13864 | in coreference resolution, 351 | 13904 | types, 405 |
| 13865 | in entity linking, 403 | 13905 | nearest-neighbor, 48, 414 |
| 13866 | in information extraction, 405 | 13906 | negation, 70, 422 |
| 13867 | mention ranking, 360 | 13907 | scope, 423 |
| 13868 | mention-pair model, 359 | 13908 | negative sampling, 336, 337, 411 |
| 13869 | meronymy, 75 | 13909 | Neo-Davidsonian event semantics, 306 |
| 13870 | method of moments, 117 | 13910 | neural attention, 371, 426, 443, 444, 460 |
| 13871 | mildly context-sensitive languages, 219 | 13911 | coarse-to-fine, 463 |
| 13872 | minibatch, 39 | 13912 | structured, 463 |
| 13873 | minimization of finite-state automata, | 13913 | neural gate, 52, 445 |
| 13874 | 196 | 13914 | neural network, 48 |
| 13875 | minimum error-rate training (MERT), | 13915 | adversarial, 113 |
| 13876 | 451 | 13916 | bidirectional recurrent, 168 |
| 13877 | minimum risk training, 451 | 13917 | convolutional, 53, 62, 185, 415 |
| 13878 | modality, 422 | 13918 | feedforward, 50 |
| 13879 | model, 8, 37 | 13919 | recurrent, 135, 415 |
| 13880 | modifier, 258 | 13920 | recursive, 250, 343, 346, 396 |
| 13881 | modus ponens, 288, 303 | 13921 | noise-contrastive estimation, 135 |
| 13882 | moment-matching, 41 | 13922 | noisy channel model, 126, 436 |
| 13883 | monomorphemic, 196 | 13923 | nominal modifier, 215 |
| 13884 | morpheme, 6, 141, 195 | 13924 | nominals, 351, 358 |
| 13885 | morphological segmentation, 159 | 13925 | non-terminals, 209 |
| 13886 | morphology, 79, 158, 194, 342, 448 | 13926 | normalization, 78 |
| 13887 | derivational, 194 | 13927 | noun phrase, 2, 212 |
| 13888 | inflectional, 177, 194, 202 | 13928 | nouns, 176 |
| 13889 | morphosyntactic, 176 | 13929 | NP-hard, 40, 409 |
| 13890 | attributes, 180 | 13930 | nucleus, in RST, 389 |
| 13891 | morphotactics, 195 | 13931 | null hypothesis, 84 |
| 13892 | multi-view learning, 108 | 13932 | null subjects, 375 |
| 13893 | multiclass classification | 13933 | numerals, 179 |
| 13894 | one-versus-all, 414 | 13934 | one-hot vector, 53 |
| 13895 | one-versus-one, 414 | 13935 | ontology, 9 |
| 13896 | multinomial | 13936 | open word classes, 176 |
| 13897 | distribution, 18 | 13937 | opinion polarity, 69 |
| 13898 | Naïve Bayes, 19 | 13938 | optimization |
| 13899 | Naïve Bayes, 17 | 13939 | batch, 37 |
| 13900 | name dictionary, 406 | 13940 | combinatorial, 8 |
| 13901 | named entity, 183 | 13941 | constrained, 31 |

- | | | | |
|-------|--|-------|---|
| 13942 | convex, 37 | 13981 | pointwise mutual information, 281, 329, 330 |
| 13943 | numerical, 8 | 13982 | in collocation identification, 344 |
| 13944 | quasi-Newton, 38 | 13983 | positive, 331 |
| 13945 | oracle, 275 | 13984 | shifted positive, 337 |
| 13946 | dynamic, 277 | 13985 | policy, 276, 365, 469 |
| 13947 | in learning to search, 366 | 13986 | policy gradient, 366 |
| 13948 | orthography, 196, 204 | 13987 | polysemy, 74 |
| 13949 | orthonormal matrix, 59 | 13988 | pooling, 63, 65, 373, 460 |
| 13950 | out-of-vocabulary words, 181 | 13989 | positional encodings, 447 |
| 13951 | outside recurrence, 237, 240 | 13990 | power law, 2 |
| 13952 | overfitting, 23, 28 | 13991 | pragmatics, 351 |
| 13953 | overgeneration, 203, 213 | 13992 | precision, 82, 478 |
| 13954 | parallel corpus, 435 | 13993 | at- <i>k</i> , 83, 398 |
| 13955 | parameters, 482 | 13994 | labeled, 230 |
| 13956 | paraphrase, 346 | 13995 | unlabeled, 230 |
| 13957 | parent annotation, 244 | 13996 | precision-recall curve, 83, 418 |
| 13958 | parsing, 209 | 13997 | predicate, 403 |
| 13959 | agenda-based, 279 | 13998 | predicative adjectives, 217 |
| 13960 | chart, 226 | 14000 | predictive likelihood, 103 |
| 13961 | easy-first, 279 | 14001 | prepositional phrase, 2, 217 |
| 13962 | graph-based, 263 | 14002 | primal form, 487 |
| 13963 | transition-based, 225 | 14003 | prior expectation, 483 |
| 13964 | part-of-speech, 6, 175 | 14004 | probabilistic models, 482 |
| 13965 | tagging, 145 | 14005 | probabilistic topic model, 5, 409 |
| 13966 | particle, 179, 217 | 14006 | probability |
| 13967 | partition, 477 | 14007 | chain rule, 477 |
| 13968 | partition function, 164 | 14008 | conditional, 34, 477, 481 |
| 13969 | passive-aggressive, 43, 487 | 14009 | density function, 481 |
| 13970 | path, in an FSA, 193 | 14010 | distribution, 480 |
| 13971 | Penn Discourse Treebank (PDTB), 385 | 14011 | joint, 16, 34, 481 |
| 13972 | Penn Treebank, 140, 159, 180, 212, 238 | 14012 | likelihood, 478 |
| 13973 | perceptron, 25 | 14013 | marginal, 481 |
| 13974 | incremental, 277, 364 | 14014 | mass function, 85, 481 |
| 13975 | multilayer, 50 | 14015 | posterior, 478 |
| 13976 | structured, 160 | 14016 | prior, 22, 36, 478 |
| 13977 | perplexity, 140 | 14017 | simplex, 18 |
| 13978 | phonology, 196 | 14018 | processes, 422 |
| 13979 | phrase, 212 | 14019 | productivity, 195 |
| 13980 | phrase-structure grammar, 212 | 14020 | projectivity, 261 |
| | | 14021 | pronominal anaphora resolution, 351 |

- | | | | |
|-------|--|-------|---|
| 14022 | pronoun, 178 | 14061 | linear, 72 |
| 14023 | reflexive, 353 | 14062 | logistic, 34 |
| 14024 | PropBank, 310 | 14063 | ridge, 72 |
| 14025 | proper nouns, 177 | 14064 | regular expression, 192 |
| 14026 | proposal distribution, 452 | 14065 | regular language, 192 |
| 14027 | propositions, 286, 287, 422 | 14066 | regularization, 23, 34 |
| 14028 | prosody, 187 | 14067 | reification (events), 305 |
| 14029 | proto-roles, 309 | 14068 | relation |
| 14030 | pumping lemma, 207 | 14069 | extraction, 277 |
| 14031 | pushdown automata, 209, 251 | 14070 | logical, 286, 295 |
| 14032 | quadratic program, 32 | 14071 | relation extraction, 411 |
| 14033 | quantifier, 289 | 14072 | relations |
| 14034 | existential, 289 | 14073 | in information extraction, 403 |
| 14035 | universal, 289 | 14074 | relative frequency estimate, 21, 126, 483 |
| 14036 | question answering, 346, 405 | 14075 | reranking, 250 |
| 14037 | cloze, 425 | 14076 | residual networks, 52 |
| 14038 | extractive, 426 | 14077 | retrofitting, 343 |
| 14039 | multiple-choice, 425 | 14078 | Rhetorical Structure Theory (RST), 389 |
| 14040 | random outcomes, 475 | 14079 | rhetorical zones, 381 |
| 14041 | random variable, 480 | 14080 | risk, 451 |
| 14042 | discrete, 480 | 14081 | roll-in, roll-out, 366, 367 |
| 14043 | indicator, 480 | 14082 | saddle point, 58, 486 |
| 14044 | ranking, 406 | 14083 | sample space, 475 |
| 14045 | loss, 406 | 14084 | satellite, in RST, 389 |
| 14046 | recall, 82, 478 | 14085 | satisfaction, 290 |
| 14047 | at- k , 398 | 14086 | scheduled sampling, 450 |
| 14048 | labeled, 230 | 14087 | schema, 403, 404, 419 |
| 14049 | unlabeled, 230 | 14088 | search error, 252, 364 |
| 14050 | receiver operating characteristic (ROC), | 14089 | segmented discourse representation |
| 14051 | 83 | 14090 | theory (SDRT), 384 |
| 14052 | rectified linear unit (ReLU), 51 | 14091 | self-attention, 446 |
| 14053 | leaky, 51 | 14092 | self-training, 108 |
| 14054 | reference arguments, 322 | 14093 | semantic concordance, 76 |
| 14055 | reference resolution, 351 | 14094 | semantic parsing, 291 |
| 14056 | reference translations, 433 | 14095 | semantic role, 306 |
| 14057 | referent, 351 | 14096 | semantic role labeling, 306, 412, 420 |
| 14058 | generic, 356 | 14097 | semantics, 285 |
| 14059 | referring expressions, 351, 381, 458 | 14098 | dynamic, 302, 384 |
| 14060 | regression, 72 | 14099 | in parsing, 242 |
| | | 14100 | lexical, 73 |

- 14101 model-theoretic, 286
14102 underspecification, 302
14103 semi-supervised learning, 105
14104 semiring, 199
14105 algebra, 172
14106 expectation, 207
14107 tropical, 173
14108 sentence compression, 465
14109 sentence fusion, 465
14110 sentence, in logic, 289
14111 sentiment analysis, 69
14112 lexicon-based, 70
14113 targeted, 71
14114 sentiment lexicon, 15
14115 sequence-to-sequence model, 442
14116 shift-reduce parsing, 251
14117 shortest-path algorithm, 197
14118 sigmoid, 49
14119 simplex, 115
14120 singular value decomposition, 59, 104
14121 truncated, 104, 330
14122 singular vectors, 60
14123 skipgram word embeddings, 335
14124 slack variables, 33
14125 slot filling, 416
14126 slots, in dialogue systems, 467
14127 smooth functions, 29
14128 smoothing, 22, 129
14129 Jeffreys-Perks, 129
14130 Kneser-Ney, 133
14131 Laplace, 22, 129
14132 Lidstone, 129
14133 softmax, 49, 134, 416
14134 hierarchical, 135, 336
14135 source language, 431
14136 spanning tree, 258
14137 sparse matrix, 330
14138 sparsity, 40
14139 spectral learning, 117
14140 speech acts, 187
14141 speech recognition, 125
14142 squashing function, 51, 135
14143 stand-off annotations, 90
14144 Stanford Natural Language Inference corpus, 346
14145 statistical significance, 84
14147 stemming, 7, 78, 195
14148 step size, 486
14149 stochastic gradient descent, 29, 38
14150 stopwords, 80
14151 string, in formal language theory, 191
14152 string-to-tree translation, 441
14153 strong compositionality criterion, 390
14154 structure induction, 170
14155 structured prediction, 15
14156 subgradient, 29, 40
14157 subjectivity detection, 71
14158 subordinating conjunctions, 178
14159 sum-product algorithm, 164
14160 summarization, 125, 393
14161 abstractive, 393, 464
14162 extractive, 393
14163 multi-document, 465
14164 of sentences, 464
14165 supersenses, 339
14166 support vector machine, 32
14167 kernel, 48, 414
14168 structured, 161
14169 support vectors, 32
14170 surface form, 203
14171 surface realization, 457
14172 synonymy, 74, 281, 325
14173 synset, 74, 370
14174 syntactic path, 313
14175 syntactic-semantic grammar, 293
14176 syntax, 175, 211, 285
14177 tagset, 176
14178 tanh activation function, 51
14179 target language, 431
14180 tense, 177
14181 terminal symbols, 209

- 14182 test set, 23, 95
 14183 test statistic, 85
 14184 text mining, 5
 14185 text planning, 457
 14186 thematic roles, 307
 14187 third axiom of probability, 476
 14188 TimeML, 421
 14189 tokenization, 77, 185
 14190 tokens and types, 19
 14191 topic segmentation, 379
 hierarchical, 381
 14192 trace, 222
 14194 training set, 17, 95
 14195 transformer architecture, 446
 14196 transition system, 269
 for context-free grammars, 251
 14198 transitive closure, 361
 14199 translation error rate (TER), 435
 14200 translation model, 126
 14201 transliteration, 449
 14202 tree-adjoining grammar, 220
 14203 tree-to-string translation, 442
 14204 tree-to-tree translation, 441
 14205 treebank, 238
 14206 trellis, 150, 205
 14207 trigrams, 24
 14208 trilexical dependencies, 248
 14209 tropical semiring, 199
 14210 true negative, 82
 14211 true positive, 82, 479
 rate, 83
 14213 truth conditions, 290
 14214 tuning set, *see* development set, 81
 14215 Turing test, 3
 14216 two-tailed test, 85
 14217 type systems, 295
 14218 type-raising, 221, 295

 14219 unary closure, 228
 14220 undercutting, in argumentation, 393, 399
 14221 underfitting, 23

 14222 underflow, 17
 14223 undergeneration, 203, 213
 14224 Universal Dependencies, 176, 257
 14225 unseen word, 169
 14226 utterances, 187

 14227 validation function, 301
 14228 validity, in logic, 290
 14229 value iteration, 469
 14230 variable, 319
 bound, 289
 free, 289
 14233 variance, 22, 87
 14234 Vauquois Pyramid, 432
 14235 verb phrase, 213
 14236 VerbNet, 308
 14237 verbs, 177
 14238 Viterbi
 algorithm, 148
 variable, 149
 14241 volition, 307

 14242 weight decay, 57
 14243 Wikification, 405
 14244 Winograd schemas, 3
 14245 word
 embeddings, 47, 53, 135, 136, 327,
 334
 embeddings, fine-tuned, 340
 embeddings, pre-trained, 339
 representations, 326
 sense disambiguation, 73
 senses, 73, 309
 tokens, 77
 14251 world model, 286
 14255 builder, 291
 14256 checker, 291

 14257 yield, 209

 14258 Zipf's law, 143