

Speech and Language Processing

An Introduction to Natural Language Processing,
Computational Linguistics, and Speech Recognition

Third Edition draft

Daniel Jurafsky
Stanford University

James H. Martin
University of Colorado at Boulder

Copyright ©2019. All rights reserved.

Draft of October 16, 2019. Comments and typos welcome!

Summary of Contents

1	Introduction.....	1
2	Regular Expressions, Text Normalization, Edit Distance.....	2
3	N-gram Language Models	30
4	Naive Bayes and Sentiment Classification.....	56
5	Logistic Regression	75
6	Vector Semantics and Embeddings	94
7	Neural Networks and Neural Language Models.....	123
8	Part-of-Speech Tagging	143
9	Sequence Processing with Recurrent Networks.....	169
10	Encoder-Decoder Models, Attention and Contextual Embeddings	191
11	Machine Translation.....	202
12	Constituency Grammars.....	203
13	Constituency Parsing.....	232
14	Statistical Constituency Parsing.....	246
15	Dependency Parsing	273
16	Logical Representations of Sentence Meaning.....	298
17	Computational Semantics and Semantic Parsing	324
18	Information Extraction	325
19	Word Senses and WordNet.....	354
20	Semantic Role Labeling.....	373
21	Lexicons for Sentiment, Affect, and Connotation	394
22	Coreference Resolution	416
23	Discourse Coherence.....	443
24	Summarization	465
25	Question Answering	466
26	Dialogue Systems and Chatbots	487
27	Phonetics.....	518
28	Speech Recognition and Synthesis	545
	Appendix	547
	A Hidden Markov Models.....	548

Contents

1	Introduction	1
2	Regular Expressions, Text Normalization, Edit Distance	2
2.1	Regular Expressions	3
2.2	Words	11
2.3	Corpora	13
2.4	Text Normalization	14
2.5	Minimum Edit Distance	23
2.6	Summary	27
	Bibliographical and Historical Notes	27
	Exercises	28
3	N-gram Language Models	30
3.1	N-Grams	31
3.2	Evaluating Language Models	36
3.3	Generalization and Zeros	38
3.4	Smoothing	42
3.5	Kneser-Ney Smoothing	46
3.6	The Web and Stupid Backoff	48
3.7	Advanced: Perplexity's Relation to Entropy	49
3.8	Summary	53
	Bibliographical and Historical Notes	53
	Exercises	54
4	Naive Bayes and Sentiment Classification	56
4.1	Naive Bayes Classifiers	58
4.2	Training the Naive Bayes Classifier	60
4.3	Worked example	62
4.4	Optimizing for Sentiment Analysis	62
4.5	Naive Bayes for other text classification tasks	64
4.6	Naive Bayes as a Language Model	65
4.7	Evaluation: Precision, Recall, F-measure	66
4.8	Test sets and Cross-validation	69
4.9	Statistical Significance Testing	69
4.10	Summary	72
	Bibliographical and Historical Notes	72
	Exercises	73
5	Logistic Regression	75
5.1	Classification: the sigmoid	76
5.2	Learning in Logistic Regression	80
5.3	The cross-entropy loss function	81
5.4	Gradient Descent	82
5.5	Regularization	87
5.6	Multinomial logistic regression	89
5.7	Interpreting models	91
5.8	Advanced: Deriving the Gradient Equation	91
5.9	Summary	92
	Bibliographical and Historical Notes	93

4 CONTENTS

Exercises	93
6 Vector Semantics and Embeddings	94
6.1 Lexical Semantics	95
6.2 Vector Semantics	98
6.3 Words and Vectors	100
6.4 Cosine for measuring similarity	103
6.5 TF-IDF: Weighing terms in the vector	105
6.6 Applications of the tf-idf vector model	107
6.7 Optional: Pointwise Mutual Information (PMI)	108
6.8 Word2vec	110
6.9 Visualizing Embeddings	115
6.10 Semantic properties of embeddings	115
6.11 Bias and Embeddings	117
6.12 Evaluating Vector Models	118
6.13 Summary	119
Bibliographical and Historical Notes	120
Exercises	122
7 Neural Networks and Neural Language Models	123
7.1 Units	124
7.2 The XOR problem	126
7.3 Feed-Forward Neural Networks	129
7.4 Training Neural Nets	132
7.5 Neural Language Models	137
7.6 Summary	141
Bibliographical and Historical Notes	141
8 Part-of-Speech Tagging	143
8.1 (Mostly) English Word Classes	143
8.2 The Penn Treebank Part-of-Speech Tagset	146
8.3 Part-of-Speech Tagging	148
8.4 HMM Part-of-Speech Tagging	149
8.5 Maximum Entropy Markov Models	159
8.6 Bidirectionality	163
8.7 Part-of-Speech Tagging for Morphological Rich Languages	164
8.8 Summary	165
Bibliographical and Historical Notes	166
Exercises	167
9 Sequence Processing with Recurrent Networks	169
9.1 Simple Recurrent Neural Networks	170
9.2 Applications of Recurrent Neural Networks	176
9.3 Deep Networks: Stacked and Bidirectional RNNs	181
9.4 Managing Context in RNNs: LSTMs and GRUs	183
9.5 Words, Subwords and Characters	187
9.6 Summary	189
Bibliographical and Historical Notes	190
10 Encoder-Decoder Models, Attention and Contextual Embeddings	191
10.1 Neural Language Models and Generation Revisited	191
10.2 Encoder-Decoder Networks	194

10.3	Attention	199
10.4	Applications of Encoder-Decoder Networks	200
10.5	Self-Attention and Transformer Networks	200
10.6	Summary	201
	Bibliographical and Historical Notes	201
11	Machine Translation	202
12	Constituency Grammars	203
12.1	Constituency	204
12.2	Context-Free Grammars	204
12.3	Some Grammar Rules for English	209
12.4	Treebanks	216
12.5	Grammar Equivalence and Normal Form	222
12.6	Lexicalized Grammars	223
12.7	Summary	228
	Bibliographical and Historical Notes	229
	Exercises	230
13	Constituency Parsing	232
13.1	Ambiguity	232
13.2	CKY Parsing: A Dynamic Programming Approach	234
13.3	Partial Parsing	240
13.4	Summary	243
	Bibliographical and Historical Notes	243
	Exercises	244
14	Statistical Constituency Parsing	246
14.1	Probabilistic Context-Free Grammars	246
14.2	Probabilistic CKY Parsing of PCFGs	251
14.3	Ways to Learn PCFG Rule Probabilities	253
14.4	Problems with PCFGs	253
14.5	Improving PCFGs by Splitting Non-Terminals	256
14.6	Probabilistic Lexicalized CFGs	258
14.7	Probabilistic CCG Parsing	262
14.8	Evaluating Parsers	268
14.9	Summary	270
	Bibliographical and Historical Notes	271
	Exercises	272
15	Dependency Parsing	273
15.1	Dependency Relations	274
15.2	Dependency Formalisms	276
15.3	Dependency Treebanks	277
15.4	Transition-Based Dependency Parsing	278
15.5	Graph-Based Dependency Parsing	289
15.6	Evaluation	294
15.7	Summary	295
	Bibliographical and Historical Notes	295
	Exercises	297
16	Logical Representations of Sentence Meaning	298

6 CONTENTS

16.1	Computational Desiderata for Representations	299
16.2	Model-Theoretic Semantics	301
16.3	First-Order Logic	304
16.4	Event and State Representations	311
16.5	Description Logics	316
16.6	Summary	321
	Bibliographical and Historical Notes	322
	Exercises	323
17	Computational Semantics and Semantic Parsing	324
18	Information Extraction	325
18.1	Named Entity Recognition	326
18.2	Relation Extraction	332
18.3	Extracting Times	342
18.4	Extracting Events and their Times	346
18.5	Template Filling	349
18.6	Summary	351
	Bibliographical and Historical Notes	352
	Exercises	353
19	Word Senses and WordNet	354
19.1	Word Senses	355
19.2	Relations Between Senses	357
19.3	WordNet: A Database of Lexical Relations	359
19.4	Word Sense Disambiguation	362
19.5	Alternate WSD algorithms and Tasks	365
19.6	Using Thesauruses to Improve Embeddings	368
19.7	Word Sense Induction	368
19.8	Summary	369
	Bibliographical and Historical Notes	370
	Exercises	371
20	Semantic Role Labeling	373
20.1	Semantic Roles	374
20.2	Diathesis Alternations	375
20.3	Semantic Roles: Problems with Thematic Roles	376
20.4	The Proposition Bank	377
20.5	FrameNet	378
20.6	Semantic Role Labeling	380
20.7	Selectional Restrictions	384
20.8	Primitive Decomposition of Predicates	389
20.9	Summary	390
	Bibliographical and Historical Notes	391
	Exercises	393
21	Lexicons for Sentiment, Affect, and Connotation	394
21.1	Defining Emotion	395
21.2	Available Sentiment and Affect Lexicons	397
21.3	Creating Affect Lexicons by Human Labeling	398
21.4	Semi-supervised Induction of Affect Lexicons	400
21.5	Supervised Learning of Word Sentiment	403

21.6	Using Lexicons for Sentiment Recognition	408
21.7	Other tasks: Personality	409
21.8	Affect Recognition	410
21.9	Lexicon-based methods for Entity-Centric Affect	411
21.10	Connotation Frames	412
21.11	Summary	413
	Bibliographical and Historical Notes	414
22	Coreference Resolution	416
22.1	Coreference Phenomena: Linguistic Background	419
22.2	Coreference Tasks and Datasets	424
22.3	Mention Detection	425
22.4	Architectures for Coreference Algorithms	428
22.5	Classifiers using hand-built features	430
22.6	A neural mention-ranking algorithm	431
22.7	Evaluation of Coreference Resolution	435
22.8	Entity Linking	436
22.9	Winograd Schema problems	437
22.10	Gender Bias in Coreference	438
22.11	Summary	440
	Bibliographical and Historical Notes	440
	Exercises	442
23	Discourse Coherence	443
23.1	Coherence Relations	445
23.2	Discourse Structure Parsing	448
23.3	Centering and Entity-Based Coherence	452
23.4	Representation learning models for local coherence	456
23.5	Global Coherence	458
23.6	Summary	461
	Bibliographical and Historical Notes	462
	Exercises	464
24	Summarization	465
25	Question Answering	466
25.1	IR-based Factoid Question Answering	467
25.2	Knowledge-based Question Answering	476
25.3	Using multiple information sources: IBM's Watson	479
25.4	Evaluation of Factoid Answers	483
	Bibliographical and Historical Notes	484
	Exercises	486
26	Dialogue Systems and Chatbots	487
26.1	Properties of Human Conversation	488
26.2	Chatbots	491
26.3	GUS: Simple Frame-based Dialogue Systems	498
26.4	The Dialogue-State Architecture	502
26.5	Evaluating Dialogue Systems	511
26.6	Dialogue System Design	512
26.7	Summary	514
	Bibliographical and Historical Notes	515

8 CONTENTS

Exercises	517
27 Phonetics	518
27.1 Speech Sounds and Phonetic Transcription	518
27.2 Articulatory Phonetics	520
27.3 Prosodic Prominence: Accent, Stress and Schwa	525
27.4 Prosodic Structure and Tune	526
27.5 Acoustic Phonetics and Signals	529
27.6 Phonetic Resources	539
27.7 Summary	542
Bibliographical and Historical Notes	543
Exercises	544
28 Speech Recognition and Synthesis	545
Appendix	547
A Hidden Markov Models	548
A.1 Markov Chains	548
A.2 The Hidden Markov Model	549
A.3 Likelihood Computation: The Forward Algorithm	551
A.4 Decoding: The Viterbi Algorithm	555
A.5 HMM Training: The Forward-Backward Algorithm	557
A.6 Summary	563
Bibliographical and Historical Notes	563
Bibliography	565
Author Index	595
Subject Index	605

CHAPTER

1

Introduction

CHAPTER

2

Regular Expressions, Text Normalization, Edit Distance

User: I am unhappy.
 ELIZA: DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY
 User: I need some help, that much seems certain.
 ELIZA: WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HELP
 User: Perhaps I could learn to get along with my mother.
 ELIZA: TELL ME MORE ABOUT YOUR FAMILY
 User: My mother takes care of me.
 ELIZA: WHO ELSE IN YOU FAMILY TAKES CARE OF YOU
 User: My father.
 ELIZA: YOUR FATHER
 User: You are like my father in some ways.

Weizenbaum (1966)

ELIZA

The dialogue above is from **ELIZA**, an early natural language processing system that could carry on a limited conversation with a user by imitating the responses of a Rogerian psychotherapist (Weizenbaum, 1966). ELIZA is a surprisingly simple program that uses pattern matching to recognize phrases like “I need X” and translate them into suitable outputs like “What would it mean to you if you got X?”. This simple technique succeeds in this domain because ELIZA doesn’t actually need to *know* anything to mimic a Rogerian psychotherapist. As Weizenbaum notes, this is one of the few dialogue genres where listeners can act as if they know nothing of the world. Eliza’s mimicry of human conversation was remarkably successful: many people who interacted with ELIZA came to believe that it really *understood* them and their problems, many continued to believe in ELIZA’s abilities even after the program’s operation was explained to them (Weizenbaum, 1976), and even today such **chatbots** are a fun diversion.

chatbots

Of course modern conversational agents are much more than a diversion; they can answer questions, book flights, or find restaurants, functions for which they rely on a much more sophisticated understanding of the user’s intent, as we will see in Chapter 26. Nonetheless, the simple pattern-based methods that powered ELIZA and other chatbots play a crucial role in natural language processing.

We’ll begin with the most important tool for describing text patterns: the **regular expression**. Regular expressions can be used to specify strings we might want to extract from a document, from transforming “I need X” in Eliza above, to defining strings like \$199 or \$24.99 for extracting tables of prices from a document.

text normalization

We’ll then turn to a set of tasks collectively called **text normalization**, in which regular expressions play an important part. Normalizing text means converting it to a more convenient, standard form. For example, most of what we are going to do with language relies on first separating out or **tokenizing** words from running text, the task of **tokenization**. English words are often separated from each other by whitespace, but whitespace is not always sufficient. *New York* and *rock ‘n’ roll* are sometimes treated as large words despite the fact that they contain spaces, while sometimes we’ll need to separate *I’m* into the two words *I* and *am*. For processing tweets or texts we’ll need to tokenize **emoticons** like :) or **hashtags** like #nlproc.

tokenization

Some languages, like Japanese, don't have spaces between words, so word tokenization becomes more difficult.

lemmatization

Another part of text normalization is **lemmatization**, the task of determining that two words have the same root, despite their surface differences. For example, the words *sang*, *sung*, and *sings* are forms of the verb *sing*. The word *sing* is the common *lemma* of these words, and a **lemmatizer** maps from all of these to *sing*. Lemmatization is essential for processing morphologically complex languages like Arabic. **Stemming** refers to a simpler version of lemmatization in which we mainly just strip suffixes from the end of the word. Text normalization also includes **sentence segmentation**: breaking up a text into individual sentences, using cues like periods or exclamation points.

stemming

sentence segmentation

Finally, we'll need to compare words and other strings. We'll introduce a metric called **edit distance** that measures how similar two strings are based on the number of edits (insertions, deletions, substitutions) it takes to change one string into the other. Edit distance is an algorithm with applications throughout language processing, from spelling correction to speech recognition to coreference resolution.

2.1 Regular Expressions

regular expression

One of the unsung successes in standardization in computer science has been the **regular expression (RE)**, a language for specifying text search strings. This practical language is used in every computer language, word processor, and text processing tools like the Unix tools grep or Emacs. Formally, a regular expression is an algebraic notation for characterizing a set of strings. They are particularly useful for searching in texts, when we have a **pattern** to search for and a **corpus** of texts to search through. A regular expression search function will search through the corpus, returning all texts that match the pattern. The corpus can be a single document or a collection. For example, the Unix command-line tool grep takes a regular expression and returns every line of the input document that matches the expression.

A search can be designed to return every match on a line, if there are more than one, or just the first match. In the following examples we generally underline the exact part of the pattern that matches the regular expression and show only the first match. We'll show regular expressions delimited by slashes but note that slashes are *not* part of the regular expressions.

Regular expressions come in many variants. We'll be describing **extended regular expressions**; different regular expression parsers may only recognize subsets of these, or treat some expressions slightly differently. Using an online regular expression tester is a handy way to test out your expressions and explore these variations.

2.1.1 Basic Regular Expression Patterns

The simplest kind of regular expression is a sequence of simple characters. To search for *woodchuck*, we type `/woodchuck/`. The expression `/Buttercup/` matches any string containing the substring *Buttercup*; grep with that expression would return the line *I'm called little Buttercup*. The search string can consist of a single character (like `!/`) or a sequence of characters (like `/urg1/`).

Regular expressions are **case sensitive**; lower case `/s/` is distinct from upper case `/S/` (`/s/` matches a lower case *s* but not an upper case *S*). This means that the pattern `/woodchucks/` will not match the string *Woodchucks*. We can solve this

RE	Example Patterns Matched
/woodchucks/	“interesting links to woodchucks and lemurs”
/a/	“Mary Ann stopped by Mona’s”
/!/	“You’ve left the burglar behind again!” said Nori

Figure 2.1 Some simple regex searches.

problem with the use of the square braces [and]. The string of characters inside the braces specifies a **disjunction** of characters to match. For example, Fig. 2.2 shows that the pattern /[wW]/ matches patterns containing either w or W.

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	“Woodchuck”
/[abc]/	‘a’, ‘b’, or ‘c’	“In uomini, in soldati”
/[1234567890]/	any digit	“plenty of 7 to 5”

Figure 2.2 The use of the brackets [] to specify a disjunction of characters.

The regular expression /[1234567890]/ specified any single digit. While such classes of characters as digits or letters are important building blocks in expressions, they can get awkward (e.g., it’s inconvenient to specify

/[ABCDEFIGHIJKLMNOPQRSTUVWXYZ]/

to mean “any capital letter”). In cases where there is a well-defined sequence associated with a set of characters, the brackets can be used with the dash (-) to specify any one character in a **range**. The pattern /2-5/ specifies any one of the characters 2, 3, 4, or 5. The pattern /[b-g]/ specifies one of the characters b, c, d, e, f, or g. Some other examples are shown in Fig. 2.3.

RE	Match	Example Patterns
/[A-Z]/	an upper case letter	“we should call it ‘Drenched Blossoms’ ”
/[a-z]/	a lower case letter	“my beans were impatient to be hoed!”
/[0-9]/	a single digit	“Chapter 1: Down the Rabbit Hole”

Figure 2.3 The use of the brackets [] plus the dash - to specify a range.

The square braces can also be used to specify what a single character *cannot* be, by use of the caret ^ . If the caret ^ is the first symbol after the open square brace [, the resulting pattern is negated. For example, the pattern /[^a]/ matches any single character (including special characters) except a. This is only true when the caret is the first symbol after the open square brace. If it occurs anywhere else, it usually stands for a caret; Fig. 2.4 shows some examples.

RE	Match (single characters)	Example Patterns
/[^A-Z]/	not an upper case letter	“Oyfn pripectchik”
/[^Ss]/	neither ‘S’ nor ‘s’	“I have no exquisite reason for’t”
/[^.]/	not a period	“our resident Djinn”
/[e^]/	either ‘e’ or ‘^’	“look up ^ now”
/a^b/	the pattern ‘a^b’	“look up a^b now”

Figure 2.4 The caret ^ for negation or just to mean ^ . See below re: the backslash for escaping the period.

How can we talk about optional elements, like an optional s in *woodchuck* and *woodchucks*? We can’t use the square brackets, because while they allow us to say “s or S”, they don’t allow us to say “s or nothing”. For this we use the question mark /?/, which means “the preceding character or nothing”, as shown in Fig. 2.5.

RE	Match	Example Patterns Matched
/woodchucks?/	woodchuck or woodchucks	“woodchuck”
/colou?r/	color or colour	“color”

Figure 2.5 The question mark ? marks optionality of the previous expression.

We can think of the question mark as meaning “zero or one instances of the previous character”. That is, it’s a way of specifying how many of something that we want, something that is very important in regular expressions. For example, consider the language of certain sheep, which consists of strings that look like the following:

baa!
baaa!
baaaa!
baaaaa!
...

Kleene *

This language consists of strings with a *b*, followed by at least two *a*’s, followed by an exclamation point. The set of operators that allows us to say things like “some number of *as*” are based on the asterisk or $*$, commonly called the **Kleene *** (generally pronounced “cleany star”). The Kleene star means “zero or more occurrences of the immediately previous character or regular expression”. So $/a^*/$ means “any string of zero or more *as*”. This will match *a* or *aaaaaa*, but it will also match *Off Minor* since the string *Off Minor* has zero *a*’s. So the regular expression for matching one or more *a* is $/aa^*/$, meaning one *a* followed by zero or more *as*. More complex patterns can also be repeated. So $/[ab]^*/$ means “zero or more *a*’s or *b*’s” (not “zero or more right square braces”). This will match strings like *aaaa* or *ababab* or *bbbb*.

For specifying multiple digits (useful for finding prices) we can extend $/[\text{0-9}]^*/$, the regular expression for a single digit. An integer (a string of digits) is thus $/[\text{0-9}][\text{0-9}]^*/$. (Why isn’t it just $/[\text{0-9}]^*/?$)

Kleene +

Sometimes it’s annoying to have to write the regular expression for digits twice, so there is a shorter way to specify “at least one” of some character. This is the **Kleene +**, which means “one or more occurrences of the immediately preceding character or regular expression”. Thus, the expression $/[\text{0-9}]^+$ is the normal way to specify “a sequence of digits”. There are thus two ways to specify the sheep language: $/baaa^*/$ or $/baa+^*/$.

One very important special character is the period ($/./$), a **wildcard** expression that matches any single character (except a carriage return), as shown in Fig. 2.6.

RE	Match	Example Matches
/beg.n/	any character between <i>beg</i> and <i>n</i>	begin, beg’n, begun

Figure 2.6 The use of the period . to specify any character.

Anchors

The wildcard is often used together with the Kleene star to mean “any string of characters”. For example, suppose we want to find any line in which a particular word, for example, *aardvark*, appears twice. We can specify this with the regular expression $/aardvark.*aardvark/$.

Anchors are special characters that anchor regular expressions to particular places in a string. The most common anchors are the caret \wedge and the dollar sign $\$$. The caret \wedge matches the start of a line. The pattern $/^\wedge The/$ matches the word *The* only at the

start of a line. Thus, the caret `^` has three uses: to match the start of a line, to indicate a negation inside of square brackets, and just to mean a caret. (What are the contexts that allow grep or Python to know which function a given caret is supposed to have?) The dollar sign `$` matches the end of a line. So the pattern `$_` is a useful pattern for matching a space at the end of a line, and `/^The dog\.$/` matches a line that contains only the phrase *The dog*. (We have to use the backslash here since we want the `.` to mean “period” and not the wildcard.)

There are also two other anchors: `\b` matches a word boundary, and `\B` matches a non-boundary. Thus, `/bthe\b/` matches the word *the* but not the word *other*. More technically, a “word” for the purposes of a regular expression is defined as any sequence of digits, underscores, or letters; this is based on the definition of “words” in programming languages. For example, `/\b99\b/` will match the string *99* in *There are 99 bottles of beer on the wall* (because *99* follows a space) but not *99* in *There are 299 bottles of beer on the wall* (since *99* follows a number). But it will match *99* in `$99` (since *99* follows a dollar sign `$`), which is not a digit, underscore, or letter).

2.1.2 Disjunction, Grouping, and Precedence

disjunction Suppose we need to search for texts about pets; perhaps we are particularly interested in cats and dogs. In such a case, we might want to search for either the string *cat* or the string *dog*. Since we can’t use the square brackets to search for “cat or dog” (why can’t we say `[/cat|dog]/?`), we need a new operator, the **disjunction** operator, also called the **pipe** symbol `|`. The pattern `/cat|dog/` matches either the string *cat* or the string *dog*.

Precedence Sometimes we need to use this disjunction operator in the midst of a larger sequence. For example, suppose I want to search for information about pet fish for my cousin David. How can I specify both *guppy* and *guppies*? We cannot simply say `/guppy|ies/`, because that would match only the strings *guppy* and *ies*. This is because sequences like *guppy* take **precedence** over the disjunction operator `|`. To make the disjunction operator apply only to a specific pattern, we need to use the parenthesis operators `(` and `)`. Enclosing a pattern in parentheses makes it act like a single character for the purposes of neighboring operators like the pipe `|` and the Kleene*. So the pattern `/gupp(y|ies)/` would specify that we meant the disjunction only to apply to the suffixes *y* and *ies*.

The parenthesis operator `(` is also useful when we are using counters like the Kleene*. Unlike the `|` operator, the Kleene* operator applies by default only to a single character, not to a whole sequence. Suppose we want to match repeated instances of a string. Perhaps we have a line that has column labels of the form *Column 1 Column 2 Column 3*. The expression `/Column[0-9]+*/` will not match any number of columns; instead, it will match a single column followed by any number of spaces! The star here applies only to the space `_` that precedes it, not to the whole sequence. With the parentheses, we could write the expression `/(Column[0-9]+_*)*/` to match the word *Column*, followed by a number and optional spaces, the whole pattern repeated zero or more times.

operator precedence This idea that one operator may take precedence over another, requiring us to sometimes use parentheses to specify what we mean, is formalized by the **operator precedence hierarchy** for regular expressions. The following table gives the order of RE operator precedence, from highest precedence to lowest precedence.

Parenthesis	()
Counters	* + ? { }
Sequences and anchors	the ^my end\$
Disjunction	

Thus, because counters have a higher precedence than sequences, /the*/ matches *theeee* but not *thethe*. Because sequences have a higher precedence than disjunction, /the|any/ matches *the* or *any* but not *thany* or *theny*.

Patterns can be ambiguous in another way. Consider the expression /[a-z]*/ when matching against the text *once upon a time*. Since /[a-z]*/ matches zero or more letters, this expression could match nothing, or just the first letter *o*, *on*, *onc*, or *once*. In these cases regular expressions always match the *largest* string they can; we say that patterns are **greedy**, expanding to cover as much of a string as they can.

greedy
non-greedy
*?
+?

There are, however, ways to enforce **non-greedy** matching, using another meaning of the ? qualifier. The operator *? is a Kleene star that matches as little text as possible. The operator +? is a Kleene plus that matches as little text as possible.

2.1.3 A Simple Example

Suppose we wanted to write a RE to find cases of the English article *the*. A simple (but incorrect) pattern might be:

/the/

One problem is that this pattern will miss the word when it begins a sentence and hence is capitalized (i.e., *The*). This might lead us to the following pattern:

/[tT]he/

But we will still incorrectly return texts with *the* embedded in other words (e.g., *other* or *theology*). So we need to specify that we want instances with a word boundary on both sides:

/\b[tT]he\b/

Suppose we wanted to do this without the use of /\b/. We might want this since /\b/ won't treat underscores and numbers as word boundaries; but we might want to find *the* in some context where it might also have underlines or numbers nearby (*the_* or *the25*). We need to specify that we want instances in which there are no alphabetic letters on either side of the *the*:

/[^a-zA-Z][tT]he[^a-zA-Z]/

But there is still one more problem with this pattern: it won't find the word *the* when it begins a line. This is because the regular expression [^a-zA-Z], which we used to avoid embedded instances of *the*, implies that there must be some single (although non-alphabetic) character before the *the*. We can avoid this by specifying that before the *the* we require *either* the beginning-of-line or a non-alphabetic character, and the same at the end of the line:

/(^|[^a-zA-Z])[tT]he([a-zA-Z]|\$)/

false positives
false negatives

The process we just went through was based on fixing two kinds of errors: **false positives**, strings that we incorrectly matched like *other* or *there*, and **false negatives**, strings that we incorrectly missed, like *The*. Addressing these two kinds of

errors comes up again and again in implementing speech and language processing systems. Reducing the overall error rate for an application thus involves two antagonistic efforts:

- Increasing **precision** (minimizing false positives)
- Increasing **recall** (minimizing false negatives)

2.1.4 A More Complex Example

Let's try out a more significant example of the power of REs. Suppose we want to build an application to help a user buy a computer on the Web. The user might want “any machine with at least 6 GHz and 500 GB of disk space for less than \$1000”. To do this kind of retrieval, we first need to be able to look for expressions like *6 GHz* or *500 GB* or *Mac* or *\$999.99*. In the rest of this section we'll work out some simple regular expressions for this task.

First, let's complete our regular expression for prices. Here's a regular expression for a dollar sign followed by a string of digits:

```
/$[0-9]+/
```

Note that the \$ character has a different function here than the end-of-line function we discussed earlier. Most regular expression parsers are smart enough to realize that \$ here doesn't mean end-of-line. (As a thought experiment, think about how regex parsers might figure out the function of \$ from the context.)

Now we just need to deal with fractions of dollars. We'll add a decimal point and two digits afterwards:

```
/$[0-9]+.\.[0-9][0-9]/
```

This pattern only allows *\$199.99* but not *\$199*. We need to make the cents optional and to make sure we're at a word boundary:

```
/(^|\W)$[0-9]+(\.\[0-9]\[0-9])?\b/
```

One last catch! This pattern allows prices like *\$199999.99* which would be far too expensive! We need to limit the dollar

```
/(^|\W)$[0-9]{0,3}(\.\[0-9]\[0-9])?\b/
```

How about disk space? We'll need to allow for optional fractions again (*5.5 GB*); note the use of ? for making the final s optional, and the of */\w*/* to mean “zero or more spaces” since there might always be extra spaces lying around:

```
/\b[0-9]+(\.\[0-9]\+)?\w*(GB|[Gg]igabytes?)\b/
```

Modifying this regular expression so that it only matches more than 500 GB is left as an exercise for the reader.

2.1.5 More Operators

Figure 2.7 shows some aliases for common ranges, which can be used mainly to save typing. Besides the Kleene * and Kleene + we can also use explicit numbers as counters, by enclosing them in curly brackets. The regular expression */\{3/* means “exactly 3 occurrences of the previous character or expression”. So */a\.\{24\}z/* will match *a* followed by 24 dots followed by *z* (but not *a* followed by 23 or 25 dots followed by a *z*).

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party_of_5
\D	[^0-9]	any non-digit	Blue_moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!
\s	[\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in_Concord

Figure 2.7 Aliases for common sets of characters.

A range of numbers can also be specified. So /{n,m}/ specifies from *n* to *m* occurrences of the previous char or expression, and /{n,}/ means at least *n* occurrences of the previous expression. REs for counting are summarized in Fig. 2.8.

RE	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{n}	<i>n</i> occurrences of the previous char or expression
{n,m}	from <i>n</i> to <i>m</i> occurrences of the previous char or expression
{n,}	at least <i>n</i> occurrences of the previous char or expression
{,m}	up to <i>m</i> occurrences of the previous char or expression

Figure 2.8 Regular expression operators for counting.

Newline Finally, certain special characters are referred to by special notation based on the backslash (\) (see Fig. 2.9). The most common of these are the **newline** character \n and the **tab** character \t. To refer to characters that are special themselves (like ., *, [, and \), precede them with a backslash, (i.e., /\./, /*//, /\[/, and /\\\/).

RE	Match	First Patterns Matched
*	an asterisk “*”	“K*A*P*L*A*N”
\.	a period “.”	“Dr. Livingston, I presume”
\?	a question mark	“Why don’t they come and lend a hand?”
\n	a newline	
\t	a tab	

Figure 2.9 Some characters that need to be backslashed.

2.1.6 Substitution, Capture Groups, and ELIZA

substitution An important use of regular expressions is in **substitutions**. For example, the substitution operator s/regexp1/pattern/ used in Python and in Unix commands like vim or sed allows a string characterized by a regular expression to be replaced by another string:

```
s/colour/color/
```

It is often useful to be able to refer to a particular subpart of the string matching the first pattern. For example, suppose we wanted to put angle brackets around all integers in a text, for example, changing *the 35 boxes* to *the <35> boxes*. We’d like a way to refer to the integer we’ve found so that we can easily add the brackets. To do this, we put parentheses (and) around the first pattern and use the **number** operator \1 in the second pattern to refer back. Here’s how it looks:

```
s/([0-9]+)/<\1>/
```

The parenthesis and number operators can also specify that a certain string or expression must occur twice in the text. For example, suppose we are looking for the pattern “the Xer they were, the Xer they will be”, where we want to constrain the two X’s to be the same string. We do this by surrounding the first X with the parenthesis operator, and replacing the second X with the number operator \1, as follows:

```
/the (.*)er they were, the \1er they will be/
```

Here the \1 will be replaced by whatever string matched the first item in parentheses. So this will match *the bigger they were, the bigger they will be* but not *the bigger they were, the faster they will be*.

capture group

register

non-capturing group

This use of parentheses to store a pattern in memory is called a **capture group**. Every time a capture group is used (i.e., parentheses surround a pattern), the resulting match is stored in a numbered **register**. If you match two different sets of parentheses, \2 means whatever matched the *second* capture group. Thus

```
/the (.*)er they (.*), the \1er we \2/
```

will match *the faster they ran, the faster we ran* but not *the faster they ran, the faster we ate*. Similarly, the third capture group is stored in \3, the fourth is \4, and so on.

Parentheses thus have a double function in regular expressions; they are used to group terms for specifying the order in which operators should apply, and they are used to capture something in a register. Occasionally we might want to use parentheses for grouping, but don’t want to capture the resulting pattern in a register. In that case we use a **non-capturing group**, which is specified by putting the commands ?: after the open paren, in the form (?: pattern).

```
/(?:some|a few) (people|cats) like some \1/
```

will match *some cats like some cats* but not *some cats like some a few*.

Substitutions and capture groups are very useful in implementing simple chatbots like ELIZA ([Weizenbaum, 1966](#)). Recall that ELIZA simulates a Rogerian psychologist by carrying on conversations like the following:

User ₁ :	Men are all alike.
ELIZA ₁ :	IN WHAT WAY
User ₂ :	They’re always bugging us about something or other.
ELIZA ₂ :	CAN YOU THINK OF A SPECIFIC EXAMPLE
User ₃ :	Well, my boyfriend made me come here.
ELIZA ₃ :	YOUR BOYFRIEND MADE YOU COME HERE
User ₄ :	He says I’m depressed much of the time.
ELIZA ₄ :	I AM SORRY TO HEAR YOU ARE DEPRESSED

ELIZA works by having a series or cascade of regular expression substitutions each of which matches and changes some part of the input lines. Input lines are first uppercased. The first substitutions then change all instances of *MY* to *YOUR*, and *I'M* to *YOU ARE*, and so on. The next set of substitutions matches and replaces other patterns in the input. Here are some examples:

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

Since multiple substitutions can apply to a given input, substitutions are assigned a rank and applied in order. Creating patterns is the topic of Exercise 2.3, and we return to the details of the ELIZA architecture in Chapter 26.

2.1.7 Lookahead Assertions

Finally, there will be times when we need to predict the future: look ahead in the text to see if some pattern matches, but not advance the match cursor, so that we can then deal with the pattern if it occurs.

lookahead zero-width These **lookahead** assertions make use of the (?) syntax that we saw in the previous section for non-capture groups. The operator (?= pattern) is true if pattern occurs, but is **zero-width**, i.e. the match pointer doesn't advance. The operator (?! pattern) only returns true if a pattern does not match, but again is zero-width and doesn't advance the cursor. Negative lookahead is commonly used when we are parsing some complex pattern but want to rule out a special case. For example suppose we want to match, at the beginning of a line, any single word that doesn't start with "Volcano". We can use negative lookahead to do this:

```
/^(?!Volcano)[A-Za-z]+/
```

2.2 Words

corpus corpora Before we talk about processing words, we need to decide what counts as a word. Let's start by looking at one particular **corpus** (plural **corpora**), a computer-readable collection of text or speech. For example the Brown corpus is a million-word collection of samples from 500 written English texts from different genres (newspaper, fiction, non-fiction, academic, etc.), assembled at Brown University in 1963–64 (Kučera and Francis, 1967). How many words are in the following Brown sentence?

He stepped out into the hall, was delighted to encounter a water brother.

This sentence has 13 words if we don't count punctuation marks as words, 15 if we count punctuation. Whether we treat period ("."), comma (","), and so on as words depends on the task. Punctuation is critical for finding boundaries of things (commas, periods, colons) and for identifying some aspects of meaning (question marks, exclamation marks, quotation marks). For some tasks, like part-of-speech tagging or parsing or speech synthesis, we sometimes treat punctuation marks as if they were separate words.

The Switchboard corpus of American English telephone conversations between strangers was collected in the early 1990s; it contains 2430 conversations averaging 6 minutes each, totaling 240 hours of speech and about 3 million words (Godfrey et al., 1992). Such corpora of spoken language don't have punctuation but do introduce other complications with regard to defining words. Let's look at one utterance from Switchboard; an **utterance** is the spoken correlate of a sentence:

I do uh main- mainly business data processing

disfluency fragment filled pause This utterance has two kinds of **disfluencies**. The broken-off word **main-** is called a **fragment**. Words like **uh** and **um** are called **fillers** or **filled pauses**. Should we consider these to be words? Again, it depends on the application. If we are building a speech transcription system, we might want to eventually strip out the disfluencies.

But we also sometimes keep disfluencies around. Disfluencies like *uh* or *um* are actually helpful in speech recognition in predicting the upcoming word, because they may signal that the speaker is restarting the clause or idea, and so for speech recognition they are treated as regular words. Because people use different disfluencies they can also be a cue to speaker identification. In fact Clark and Fox Tree (2002) showed that *uh* and *um* have different meanings. What do you think they are?

Are capitalized tokens like *They* and uncapitalized tokens like *they* the same word? These are lumped together in some tasks (speech recognition), while for part-of-speech or named-entity tagging, capitalization is a useful feature and is retained.

How about inflected forms like *cats* versus *cat*? These two words have the same lemma cat but are different wordforms. A lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense. The word-form is the full inflected or derived form of the word. For morphologically complex languages like Arabic, we often need to deal with lemmatization. For many tasks in English, however, wordforms are sufficient.

How many words are there in English? To answer this question we need to distinguish two ways of talking about words. Types are the number of distinct words in a corpus; if the set of words in the vocabulary is V , the number of types is the vocabulary size $|V|$. Tokens are the total number N of running words. If we ignore punctuation, the following Brown sentence has 16 tokens and 14 types:

They picnicked by the pool, then lay back on the grass and looked at the stars.

When we speak about the number of words in the language, we are generally referring to word types.

Corpus	Tokens = N	Types = $ V $
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13 million

Figure 2.10 Rough numbers of types and tokens for some English language corpora. The largest, the Google N-grams corpus, contains 13 million types, but this count only includes types appearing 40 or more times, so the true number would be much larger.

Herdan's Law
Heaps' Law

Fig. 2.10 shows the rough numbers of types and tokens computed from some popular English corpora. The larger the corpora we look at, the more word types we find, and in fact this relationship between the number of types $|V|$ and number of tokens N is called Herdan's Law (Herdan, 1960) or Heaps' Law (Heaps, 1978) after its discoverers (in linguistics and information retrieval respectively). It is shown in Eq. 2.1, where k and β are positive constants, and $0 < \beta < 1$.

$$|V| = kN^\beta \quad (2.1)$$

The value of β depends on the corpus size and the genre, but at least for the large corpora in Fig. 2.10, β ranges from .67 to .75. Roughly then we can say that the vocabulary size for a text goes up significantly faster than the square root of its length in words.

Another measure of the number of words in the language is the number of lemmas instead of wordform types. Dictionaries can help in giving lemma counts; dictionary entries or boldface forms are a very rough upper bound on the number of

lemmas (since some lemmas have multiple boldface forms). The 1989 edition of the Oxford English Dictionary had 615,000 entries.

2.3 Corpora

Words don't appear out of nowhere. Any particular piece of text that we study is produced by one or more specific speakers or writers, in a specific dialect of a specific language, at a specific time, in a specific place, for a specific function.

Perhaps the most important dimension of variation is the language. NLP algorithms are most useful when they apply across many languages. The world has 7097 languages at the time of this writing, according to the online Ethnologue catalog (Simons and Fennig, 2018). Most NLP tools tend to be developed for the official languages of large industrialized nations (Chinese, English, Spanish, Arabic, etc.), but we don't want to limit tools to just these few languages. Furthermore, most languages also have multiple varieties, such as dialects spoken in different regions or by different social groups. Thus, for example, if we're processing text in African American Vernacular English (AAVE), a dialect spoken by millions of people in the United States, it's important to make use of NLP tools that function with that dialect.

AAVE

Twitter posts written in AAVE make use of constructions like *iont* (*I don't* in Standard American English (SAE)), or *talmbout* corresponding to SAE *talking about*, both examples that influence word segmentation (Blodgett et al. 2016, Jones 2015).

SAE

It's also quite common for speakers or writers to use multiple languages in a single communicative act, a phenomenon called **code switching**. Code switching is enormously common across the world; here are examples showing Spanish and (transliterated) Hindi code switching with English (Solorio et al. 2014, Jurgens et al. 2017):

- (2.2) Por primera vez veo a @username actually being hateful! it was beautiful:)
[*For the first time I get to see @username actually being hateful! it was beautiful:)*]
- (2.3) dost tha or ra- hega ... dont wory ... but dherya rakhe
[*"he was and will remain a friend ... don't worry ... but have faith"*]

Another dimension of variation is the genre. The text that our algorithms must process might come from newswire, fiction or non-fiction books, scientific articles, Wikipedia, or religious texts. It might come from spoken genres like telephone conversations, business meetings, police body-worn cameras, medical interviews, or transcripts of television shows or movies. It might come from work situations like doctors' notes, legal text, or parliamentary or congressional proceedings.

Text also reflects the demographic characteristics of the writer (or speaker): their age, gender, race, socioeconomic class can all influence the linguistic properties of the text we are processing.

And finally, time matters too. Language changes over time, and for some languages we have good corpora of texts from different historical periods.

Because language is so situated, when developing computational models for language processing, it's important to consider who produced the language, in what context, for what purpose, and make sure that the models are fit to the data.

code switching

2.4 Text Normalization

Before almost any natural language processing of a text, the text has to be normalized. At least three tasks are commonly applied as part of any normalization process:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

In the next sections we walk through each of these tasks.

2.4.1 Unix Tools for Crude Tokenization and Normalization

Let's begin with an easy, if somewhat naive version of word tokenization and normalization (and frequency computation) that can be accomplished for English solely in a single UNIX command-line, inspired by [Church \(1994\)](#). We'll make use of some Unix commands: `tr`, used to systematically change particular characters in the input; `sort`, which sorts input lines in alphabetical order; and `uniq`, which collapses and counts adjacent identical lines.

For example let's begin with the 'complete words' of Shakespeare in one textfile, `sh.txt`. We can use `tr` to tokenize the words by changing every sequence of non-alphabetic characters to a newline ('A-Za-z' means alphabetic, the `-c` option complements to non-alphabet, and the `-s` option squeezes all sequences into a single character):

```
tr -sc 'A-Za-z' '\n' < sh.txt
```

The output of this command will be:

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

Now that there is one word per line, we can sort the lines, and pass them to `uniq -c` which will collapse and count them:

```
tr -sc 'A-Za-z' '\n' < sh.txt | sort | uniq -c
```

with the following output:

```
1945 A
72 AARON
19 ABBESS
25 Aaron
6 Abate
1 Abates
5 Abbess
6 Abbey
```

3 Abbot

...

Alternatively, we can collapse all the upper case to lower case:

```
tr -sc 'A-Za-z' '\n' < sh.txt | tr A-Z a-z | sort | uniq -c
```

whose output is

```
14725 a
 97 aaron
  1 abaissiez
10 abandon
  2 abandoned
  2 abase
  1 abash
14 abate
  3 abated
  3 abatement
...
```

Now we can sort again to find the frequent words. The `-n` option to `sort` means to sort numerically rather than alphabetically, and the `-r` option means to sort in reverse order (highest-to-lowest):

```
tr -sc 'A-Za-z' '\n' < sh.txt | tr A-Z a-z | sort | uniq -c | sort -n -r
```

The results show that the most frequent words in Shakespeare, as in any other corpus, are the short **function words** like articles, pronouns, prepositions:

```
27378 the
26084 and
22538 i
19771 to
17481 of
14725 a
13826 you
12489 my
11318 that
11112 in
...
```

Unix tools of this sort can be very handy in building quick word count statistics for any corpus.

2.4.2 Word Tokenization

The simple UNIX tools above were fine for getting rough word statistics but more sophisticated algorithms are generally necessary for **tokenization**, the task of segmenting running text into words.

While the Unix command sequence just removed all the numbers and punctuation, for most NLP applications we'll need to keep these in our tokenization. We often want to break off punctuation as a separate token; commas are a useful piece of information for parsers, periods help indicate sentence boundaries. But we'll often want to keep the punctuation that occurs word internally, in examples like *m.p.h.*, *Ph.D.*, *AT&T*, *cap'n*. Special characters and numbers will need to be kept in prices

(\$45.55) and dates (01/02/06); we don't want to segment that price into separate tokens of "45" and "55". And there are URLs (<http://www.stanford.edu>), Twitter hashtags (#nlp), or email addresses (someone@cs.colorado.edu).

Number expressions introduce other complications as well; while commas normally appear at word boundaries, commas are used inside numbers in English, every three digits: 555,500.50. Languages, and hence tokenization requirements, differ on this; many continental European languages like Spanish, French, and German, by contrast, use a comma to mark the decimal point, and spaces (or sometimes periods) where English puts commas, for example, 555 500,50.

clitic

A tokenizer can also be used to expand **clitic** contractions that are marked by apostrophes, for example, converting what're to the two tokens what are, and we're to we are. A clitic is a part of a word that can't stand on its own, and can only occur when it is attached to another word. Some such contractions occur in other alphabetic languages, including articles and pronouns in French (j'ai, l'homme).

Depending on the application, tokenization algorithms may also tokenize multiword expressions like New York or rock 'n' roll as a single token, which requires a multiword expression dictionary of some sort. Tokenization is thus intimately tied up with **named entity detection**, the task of detecting names, dates, and organizations (Chapter 18).

Penn Treebank tokenization

One commonly used tokenization standard is known as the **Penn Treebank tokenization** standard, used for the parsed corpora (treebanks) released by the Linguistic Data Consortium (LDC), the source of many useful datasets. This standard separates out clitics (doesn't becomes does plus n't), keeps hyphenated words together, and separates out all punctuation (to save space we're showing visible spaces ' ' between tokens, although newlines is a more common output):

Input: "The San Francisco-based restaurant," they said,
"doesn't charge \$10".
Output: "TheSanFrancisco-basedrestaurant,"theysaid,
"doesn'tcharge\$10".

In practice, since tokenization needs to be run before any other language processing, it needs to be very fast. The standard method for tokenization is therefore to use deterministic algorithms based on regular expressions compiled into very efficient finite state automata. For example, Fig. 2.11 shows an example of a basic regular expression that can be used to tokenize with the `nltk.regexp_tokenize` function of the Python-based Natural Language Toolkit (NLTK) (Bird et al. 2009; <http://www.nltk.org>).

Carefully designed deterministic algorithms can deal with the ambiguities that arise, such as the fact that the apostrophe needs to be tokenized differently when used as a genitive marker (as in *the book's cover*), a quotative as in '*The other class*', *she said*, or in clitics like *they're*.

Word tokenization is more complex in languages like written Chinese, Japanese, and Thai, which do not use spaces to mark potential word-boundaries.

hanzi

In Chinese, for example, words are composed of characters (called **hanzi** in Chinese). Each character generally represents a single unit of meaning (called a **morpheme**) and is pronounceable as a single syllable. Words are about 2.4 characters long on average. But deciding what counts as a word in Chinese is complex. For example, consider the following sentence:

(2.4) 姚明进入总决赛
"Yao Ming reaches the finals"

```

>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)      # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...     | \w+(-\w+)*       # words with optional internal hyphens
...     | \$?\d+(\.\d+)?%? # currency and percentages, e.g. $12.40, 82%
...     | \.\.\.            # ellipsis
...     | [][,;]?():-_`]  # these are separate tokens; includes ], [
...     ,,
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']

```

Figure 2.11 A python trace of regular expression tokenization in the NLTK (Bird et al., 2009) Python-based natural language processing toolkit, commented for readability; the (?x) verbose flag tells Python to strip comments and whitespace. Figure from Chapter 3 of Bird et al. (2009).

As Chen et al. (2017) point out, this could be treated as 3 words ('Chinese Treebank' segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words ('Peking University' segmentation):

(2.6) 姚 明 进 入 总 决 赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

In fact, for most Chinese NLP tasks it turns out to work better to take characters rather than words as input, since characters are at a reasonable semantic level for most applications, and since most word standards result in a huge vocabulary with large numbers of very rare words (Li et al., 2019).

word
segmentation

However, for Japanese and Thai the character is too small a unit, and so algorithms for **word segmentation** are required. These can also be useful for Chinese in the rare situations where word rather than character boundaries are required. The standard segmentation algorithms for these languages use neural **sequence models** trained via supervised machine learning on hand-segmented training sets; we'll introduce sequence models in Chapter 8.

2.4.3 Byte-Pair Encoding for Tokenization

There is a third option to tokenizing text input. Instead of defining tokens as words (defined by spaces in orthographies that have spaces, or more complex algorithms), or as characters (as in Chinese), we can use our data to automatically tell us what size tokens should be. Perhaps sometimes we might want tokens that are space-delimited words (like *spinach*) other times it's useful to have tokens that are larger than words (like *New York Times*), and sometimes smaller than words (like the morphemes *-est* or *-er*. A morpheme is the smallest meaning-bearing unit of a language; for example the word *unlikeliest* has the morphemes *un-*, *likely*, and *-est*; we'll return to this on page 21.

subword

One reason it's helpful to have **subword** tokens is to deal with unknown words.

Unknown words are particularly relevant for machine learning systems. As we will see in the next chapter, machine learning systems often learn some facts about words in one corpus (a **training** corpus) and then use these facts to make decisions about a separate **test** corpus and its words. Thus if our training corpus contains, say the words *low*, and *lowest*, but not *lower*, but then the word *lower* appears in our test corpus, our system will not know what to do with it.

A solution to this problem is to use a kind of tokenization in which most tokens are words, but some tokens are frequent morphemes or other subwords like *-er*, so that an unseen word can be represented by combining the parts.

BPE The simplest such algorithm is **byte-pair encoding**, or **BPE** ([Sennrich et al., 2016](#)). Byte-pair encoding is based on a method for text compression ([Gage, 1994](#)), but here we use it for tokenization instead. The intuition of the algorithm is to iteratively merge frequent pairs of characters,

The algorithm begins with the set of symbols equal to the set of characters. Each word is represented as a sequence of characters plus a special end-of-word symbol *_*. At each step of the algorithm, we count the number of symbol pairs, find the most frequent pair ('A', 'B'), and replace it with the new merged symbol ('AB'). We continue to count and merge, creating new longer and longer character strings, until we've done k merges; k is a parameter of the algorithm. The resulting symbol set will consist of the original set of characters plus k new symbols.

The algorithm is run inside words (we don't merge across word boundaries). For this reason, the algorithm can take as input a dictionary of words together with counts. Consider the following tiny input dictionary with counts for each word, which would have the starting vocabulary of 11 letters:

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w
2	l o w e s t _	
6	n e w e r _	
3	w i d e r _	
2	n e w _	

We first count all pairs of symbols: the most frequent is the pair *r_* because it occurs in *newer* (frequency of 6) and *wider* (frequency of 3) for a total of 9 occurrences. We then merge these symbols, treating *r_* as one symbol, and count again:

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Now the most frequent pair is *e r_*, which we merge; our system has learned that there should be a token for word-final *er*, represented as *er_*:

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_
2	l o w e s t _	
6	n e w e r_	
3	w i d e r_	
2	n e w _	

Next *e w* (total count of 8) get merged to *ew*:

	dictionary	vocabulary
5	l o w _	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew
2	l o w e s t _	
6	n ew er_	
3	w i d er_	
2	n ew _	

If we continue, the next merges are:

Merge	Current Vocabulary
(n, ew)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new
(l, o')	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_
(low, __)	_, d, e, i, l, n, o, r, s, t, w, r_, er_, ew, new, lo, low, newer_, low_

When we need to tokenize a test sentence, we just run the merges we have learned, greedily, in the order we learned them, on the test data. (Thus the frequencies in the test data don't play a role, just the frequencies in the training data). So first we segment each test sentence word into characters. Then we apply the first rule: replace every instance of `r _` in the test corpus with `r_`, and then the second rule: replace every instance of `e r_` in the test corpus with `er_`, and so on. By the end, if the test corpus contained the word `n e w e r _`, it would be tokenized as a full word. But a new (unknown) word like `l o w e r _` would be merged into the two tokens `low er_`.

Of course in real algorithms BPE is run with many thousands of merges on a very large input dictionary. The result is that most words will be represented as full symbols, and only the very rare words (and unknown words) will have to be represented by their parts. The full BPE learning algorithm is given in Fig. 2.12.

Wordpiece and Greedy Tokenization

There are some alternatives to byte pair encoding for inducing tokens. Like the BPE algorithm, the **wordpiece** algorithm starts with some simple tokenization (such as by whitespace) into rough words, and then breaks those rough word tokens into subword tokens. The **wordpiece** model differs from BPE only in that the special word-boundary token `_` appears at the beginning of words rather than at the end, and in the way it merges pairs. Rather than merging the pairs that are most *frequent*, wordpiece instead merges the pairs that minimizes the language model likelihood of the training data. We'll introduce these concepts in the next chapter, but to simplify, the wordpiece model chooses the two tokens to combine that would give the training corpus the highest probability (Wu et al., 2016).

In the wordpiece segmenter used in BERT (Devlin et al., 2019), like other wordpiece variants, an input sentence or string is first split by some simple basic tokenizer (like whitespace) into a series of rough word tokens. But then instead of using a word boundary token, word-initial subwords are distinguished from those that do not start words by marking internal subwords with special symbols `##`, so that we might split `unaffable` into `["un", "\#\#aff", "\#\#\#able"]`. Then each word token string is tokenized using a greedy longest-match-first algorithm. This is different than the decoding algorithm we introduced for BPE, which runs the merges on the test sentence in the same order they were learned from the training set.

Greedy longest-match-first decoding is sometimes called **maximum matching** or **MaxMatch**. The maximum matching algorithm (Fig. 2.13) is given a vocabulary (a learned list of wordpiece tokens) and a string and starts by pointing at the

```

import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i], symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(''.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l_o_w_</w>': 5, 'l_o_w_e_s_t_</w>': 2,
         'n_e_w_e_r_</w>': 6, 'w_i_d_e_r_u_</w>': 3, 'n_e_w_</w>': 2}
num_merges = 8

for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)

```

Figure 2.12 Python code for BPE learning algorithm from Sennrich et al. (2016).

beginning of a string. It chooses the longest token in the wordpiece vocabulary that matches the input at the current position, and moves the pointer past that word in the string. The algorithm is then applied again starting from the new pointer position.

```

function MAXMATCH(string,dictionary) returns list of tokens T
    if string is empty
        return empty list
    for i  $\leftarrow$  length(sentence) downto 1
        firstword = first i chars of sentence
        remainder = rest of sentence
        if InDictionary(firstword, dictionary)
            return list(firstword, MaxMatch(remainder,dictionary) )

```

Figure 2.13 The MaxMatch (or ‘greedy longest-first’) algorithm for word tokenization using wordpiece or other vocabularies. Assumes that all strings can be successfully tokenized with the given dictionary.

Thus given the token `intention` and the dictionary:

`["in", "tent", "intent", "#tent", "#tention", "#tion", "#ion"]`
the BERT tokenizer would choose `intent` (because it is longer than `in`, and then `#ion` to complete the string, resulting in the tokenization `["intent" "#ion"]`).
The BERT tokenizer applied to the string `unwanted running` will produce:

(2.8) `["un", "#want", "#ed", "runn", "#ing"]`

SentencePiece

Another tokenization algorithm is called **SentencePiece** (Kudo and Richardson,

2018). BPE and wordpiece both assume that we already have some initial tokenization of words (such as by spaces, or from some initial dictionary) and so we never tried to induce word parts across spaces. By contrast, the SentencePiece model works from raw text; even whitespace is handled as a normal symbol. Thus it doesn't need an initial tokenization or word-list, and can be used in languages like Chinese or Japanese that don't have spaces.

2.4.4 Word Normalization, Lemmatization and Stemming

normalization

Word **normalization** is the task of putting words/tokens in a standard format, choosing a single normal form for words with multiple forms like USA and US or uh-huh and uhhuh. This standardization may be valuable, despite the spelling information that is lost in the normalization process. For information retrieval or information extraction about the US, we might want see information from documents whether they mention the US or the USA.

case folding

Case folding is another kind of normalization. Mapping everything to lower case means that *Woodchuck* and *woodchuck* are represented identically, which is very helpful for generalization in many tasks, such as information retrieval or speech recognition. For sentiment analysis and other text classification tasks, information extraction, and machine translation, by contrast, case can be quite helpful and case folding is generally not done. This is because maintaining the difference between, for example, US the country and us the pronoun can outweigh the advantage in generalization that case folding would have provided for other words.

For many natural language processing situations we also want two morphologically different forms of a word to behave similarly. For example in web search, someone may type the string *woodchucks* but a useful system might want to also return pages that mention *woodchuck* with no *s*. This is especially common in morphologically complex languages like Russian, where for example the word *Moscow* has different endings in the phrases *Moscow*, *of Moscow*, *to Moscow*, and so on.

Lemmatization is the task of determining that two words have the same root, despite their surface differences. The words *am*, *are*, and *is* have the shared lemma *be*; the words *dinner* and *dinners* both have the lemma *dinner*. Lemmatizing each of these forms to the same lemma will let us find all mentions of words in Russian like *Moscow*. The lemmatized form of a sentence like *He is reading detective stories* would thus be *He be read detective story*.

morpheme

How is lemmatization done? The most sophisticated methods for lemmatization involve complete **morphological parsing** of the word. **Morphology** is the study of the way words are built up from smaller meaning-bearing units called **morphemes**. Two broad classes of morphemes can be distinguished: **stems**—the central morpheme of the word, supplying the main meaning—and **affixes**—adding “additional” meanings of various kinds. So, for example, the word *fox* consists of one morpheme (the morpheme *fox*) and the word *cats* consists of two: the morpheme *cat* and the morpheme *-s*. A morphological parser takes a word like *cats* and parses it into the two morphemes *cat* and *s*, or a Spanish word like *amar* (‘if in the future they would love’) into the morphemes *amar* ‘to love’, *3PL*, and *future subjunctive*.

The Porter Stemmer

stemming Porter stemmer

Lemmatization algorithms can be complex. For this reason we sometimes make use of a simpler but cruder method, which mainly consists of chopping off word-final affixes. This naive version of morphological analysis is called **stemming**. One of the most widely used stemming algorithms is the [Porter \(1980\)](#). The Porter stemmer

applied to the following paragraph:

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.

produces the following stemmed output:

Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

cascade The algorithm is based on series of rewrite rules run in series, as a **cascade**, in which the output of each pass is fed as input to the next pass; here is a sampling of the rules:

ATIONAL → ATE (e.g., relational → relate)

ING → ε if stem contains vowel (e.g., motoring → motor)

SSES → SS (e.g., grasses → grass)

Detailed rule lists for the Porter stemmer, as well as code (in Java, Python, etc.) can be found on Martin Porter’s homepage; see also the original paper ([Porter, 1980](#)).

Simple stemmers can be useful in cases where we need to collapse across different variants of the same lemma. Nonetheless, they do tend to commit errors of both over- and under-generalizing, as shown in the table below ([Krovetz, 1993](#)):

Errors of Commission		Errors of Omission	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity

2.4.5 Sentence Segmentation

Sentence segmentation

Sentence segmentation is another important step in text processing. The most useful cues for segmenting a text into sentences are punctuation, like periods, question marks, and exclamation points. Question marks and exclamation points are relatively unambiguous markers of sentence boundaries. Periods, on the other hand, are more ambiguous. The period character “.” is ambiguous between a sentence boundary marker and a marker of abbreviations like *Mr.* or *Inc.* The previous sentence that you just read showed an even more complex case of this ambiguity, in which the final period of *Inc.* marked both an abbreviation and the sentence boundary marker. For this reason, sentence tokenization and word tokenization may be addressed jointly.

In general, sentence tokenization methods work by first deciding (based on rules or machine learning) whether a period is part of the word or is a sentence-boundary marker. An abbreviation dictionary can help determine whether the period is part of a commonly used abbreviation; the dictionaries can be hand-built or machine-learned ([Kiss and Strunk, 2006](#)), as can the final sentence splitter. In the Stanford CoreNLP toolkit ([Manning et al., 2014](#)), for example sentence splitting is rule-based, a deterministic consequence of tokenization; a sentence ends when a sentence-ending punctuation (., !, or ?) is not already grouped with other characters into a token (such as for an abbreviation or number), optionally followed by additional final quotes or brackets.

2.5 Minimum Edit Distance

Much of natural language processing is concerned with measuring how similar two strings are. For example in spelling correction, the user typed some erroneous string—let's say *graffe*—and we want to know what the user meant. The user probably intended a word that is similar to *graffe*. Among candidate similar words, the word *giraffe*, which differs by only one letter from *graffe*, seems intuitively to be more similar than, say *grail* or *graf*, which differ in more letters. Another example comes from **coreference**, the task of deciding whether two strings such as the following refer to the same entity:

Stanford President John Hennessy
Stanford University President John Hennessy

Again, the fact that these two strings are very similar (differing by only one word) seems like useful evidence for deciding that they might be coreferent.

minimum edit distance

alignment

Edit distance gives us a way to quantify both of these intuitions about string similarity. More formally, the **minimum edit distance** between two strings is defined as the minimum number of editing operations (operations like insertion, deletion, substitution) needed to transform one string into another.

The gap between *intention* and *execution*, for example, is 5 (delete an *i*, substitute *e* for *n*, substitute *x* for *t*, insert *c*, substitute *u* for *n*). It's much easier to see this by looking at the most important visualization for string distances, an **alignment** between the two strings, shown in Fig. 2.14. Given two sequences, an **alignment** is a correspondence between substrings of the two sequences. Thus, we say *I* aligns with the empty string, *N* with *E*, and so on. Beneath the aligned strings is another representation; a series of symbols expressing an **operation list** for converting the top string into the bottom string: **d** for deletion, **s** for substitution, **i** for insertion.

<table style="width: 100%; border-collapse: collapse;"> <tr> <td style="text-align: right; padding-right: 10px;">I N T E * N T I O N</td><td style="border-left: 1px solid black; padding-left: 10px;"></td></tr> <tr> <td style="text-align: right; padding-right: 10px;"> </td><td style="border-left: 1px solid black; padding-left: 10px;"></td></tr> <tr> <td style="text-align: right; padding-right: 10px;">* E X E C U T I O N</td><td style="border-left: 1px solid black; padding-left: 10px;"></td></tr> <tr> <td style="text-align: right; padding-right: 10px;">d s s i s</td><td style="border-left: 1px solid black; padding-left: 10px;"></td></tr> </table>		I N T E * N T I O N				* E X E C U T I O N		d s s i s	
I N T E * N T I O N									
* E X E C U T I O N									
d s s i s									

Figure 2.14 Representing the minimum edit distance between two strings as an **alignment**. The final row gives the operation list for converting the top string into the bottom string: **d** for deletion, **s** for substitution, **i** for insertion.

We can also assign a particular cost or weight to each of these operations. The **Levenshtein** distance between two sequences is the simplest weighting factor in which each of the three operations has a cost of 1 (Levenshtein, 1966)—we assume that the substitution of a letter for itself, for example, *t* for *t*, has zero cost. The Levenshtein distance between *intention* and *execution* is 5. Levenshtein also proposed an alternative version of his metric in which each insertion or deletion has a cost of 1 and substitutions are not allowed. (This is equivalent to allowing substitution, but giving each substitution a cost of 2 since any substitution can be represented by one insertion and one deletion). Using this version, the Levenshtein distance between *intention* and *execution* is 8.

2.5.1 The Minimum Edit Distance Algorithm

How do we find the minimum edit distance? We can think of this as a search task, in which we are searching for the shortest path—a sequence of edits—from one string to another.

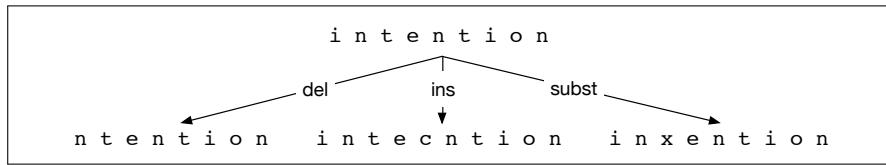


Figure 2.15 Finding the edit distance viewed as a search problem

dynamic
programming

The space of all possible edits is enormous, so we can't search naively. However, lots of distinct edit paths will end up in the same state (string), so rather than recomputing all those paths, we could just remember the shortest path to a state each time we saw it. We can do this by using **dynamic programming**. Dynamic programming is the name for a class of algorithms, first introduced by [Bellman \(1957\)](#), that apply a table-driven method to solve problems by combining solutions to sub-problems. Some of the most commonly used algorithms in natural language processing make use of dynamic programming, such as the **Viterbi** algorithm (Chapter 8) and the **CKY** algorithm for parsing (Chapter 13).

The intuition of a dynamic programming problem is that a large problem can be solved by properly combining the solutions to various sub-problems. Consider the shortest path of transformed words that represents the minimum edit distance between the strings *intention* and *execution* shown in Fig. 2.16.

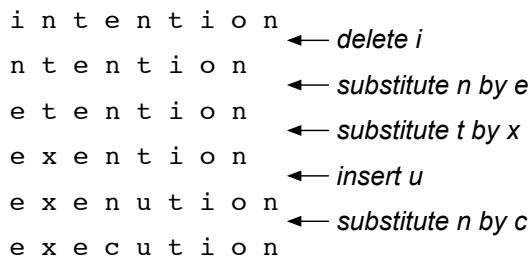


Figure 2.16 Path from *intention* to *execution*.

minimum edit
distance

Imagine some string (perhaps it is *exentio*) that is in this optimal path (whatever it is). The intuition of dynamic programming is that if *exentio* is in the optimal operation list, then the optimal sequence must also include the optimal path from *intention* to *exentio*. Why? If there were a shorter path from *intention* to *exentio*, then we could use it instead, resulting in a shorter overall path, and the optimal sequence wouldn't be optimal, thus leading to a contradiction.

The **minimum edit distance** algorithm was named by [Wagner and Fischer \(1974\)](#) but independently discovered by many people (see the Historical Notes section of Chapter 8).

Let's first define the minimum edit distance between two strings. Given two strings, the source string X of length n , and target string Y of length m , we'll define $D[i, j]$ as the edit distance between $X[1..i]$ and $Y[1..j]$, i.e., the first i characters of X and the first j characters of Y . The edit distance between X and Y is thus $D[n, m]$.

We'll use dynamic programming to compute $D[n, m]$ bottom up, combining solutions to subproblems. In the base case, with a source substring of length i but an empty target string, going from i characters to 0 requires i deletes. With a target substring of length j but an empty source going from 0 characters to j characters requires j inserts. Having computed $D[i, j]$ for small i, j we then compute larger $D[i, j]$ based on previously computed smaller values. The value of $D[i, j]$ is computed by taking the minimum of the three possible paths through the matrix which arrive there:

$$D[i, j] = \min \begin{cases} D[i - 1, j] + \text{del-cost}(\text{source}[i]) \\ D[i, j - 1] + \text{ins-cost}(\text{target}[j]) \\ D[i - 1, j - 1] + \text{sub-cost}(\text{source}[i], \text{target}[j]) \end{cases}$$

If we assume the version of Levenshtein distance in which the insertions and deletions each have a cost of 1 ($\text{ins-cost}(\cdot) = \text{del-cost}(\cdot) = 1$), and substitutions have a cost of 2 (except substitution of identical letters have zero cost), the computation for $D[i, j]$ becomes:

$$D[i, j] = \min \begin{cases} D[i - 1, j] + 1 \\ D[i, j - 1] + 1 \\ D[i - 1, j - 1] + \begin{cases} 2; & \text{if } \text{source}[i] \neq \text{target}[j] \\ 0; & \text{if } \text{source}[i] = \text{target}[j] \end{cases} \end{cases} \quad (2.9)$$

The algorithm is summarized in Fig. 2.17; Fig. 2.18 shows the results of applying the algorithm to the distance between *intention* and *execution* with the version of Levenshtein in Eq. 2.9.

Knowing the minimum edit distance is useful for algorithms like finding potential spelling error corrections. But the edit distance algorithm is important in another way; with a small change, it can also provide the minimum cost **alignment** between two strings. Aligning two strings is useful throughout speech and language processing. In speech recognition, minimum edit distance alignment is used to compute the word error rate (Chapter 28). Alignment plays a role in machine translation, in which sentences in a parallel corpus (a corpus with a text in two languages) need to be matched to each other.

To extend the edit distance algorithm to produce an alignment, we can start by visualizing an alignment as a path through the edit distance matrix. Figure 2.19 shows this path with the boldfaced cell. Each boldfaced cell represents an alignment of a pair of letters in the two strings. If two boldfaced cells occur in the same row, there will be an insertion in going from the source to the target; two boldfaced cells in the same column indicate a deletion.

backtrace

Figure 2.19 also shows the intuition of how to compute this alignment path. The computation proceeds in two steps. In the first step, we augment the minimum edit distance algorithm to store backpointers in each cell. The backpointer from a cell points to the previous cell (or cells) that we came from in entering the current cell. We've shown a schematic of these backpointers in Fig. 2.19. Some cells have multiple backpointers because the minimum extension could have come from multiple previous cells. In the second step, we perform a **backtrace**. In a backtrace, we start from the last cell (at the final row and column), and follow the pointers back through the dynamic programming matrix. Each complete path between the final cell and the initial cell is a minimum distance alignment. Exercise 2.7 asks you to modify the minimum edit distance algorithm to store the pointers and compute the backtrace to output an alignment.

```

function MIN-EDIT-DISTANCE(source, target) returns min-distance

n  $\leftarrow$  LENGTH(source)
m  $\leftarrow$  LENGTH(target)
Create a distance matrix distance[n+1, m+1]

# Initialization: the zeroth row and column is the distance from the empty string
D[0,0] = 0
for each row i from 1 to n do
    D[i,0]  $\leftarrow$  D[i-1,0] + del-cost(source[i])
for each column j from 1 to m do
    D[0,j]  $\leftarrow$  D[0,j-1] + ins-cost(target[j])

# Recurrence relation:
for each row i from 1 to n do
    for each column j from 1 to m do
        D[i,j]  $\leftarrow$  MIN( D[i-1,j] + del-cost(source[i]),
                           D[i-1,j-1] + sub-cost(source[i], target[j]),
                           D[i,j-1] + ins-cost(target[j]))
# Termination
return D[n,m]

```

Figure 2.17 The minimum edit distance algorithm, an example of the class of dynamic programming algorithms. The various costs can either be fixed (e.g., $\forall x, \text{ins-cost}(x) = 1$) or can be specific to the letter (to model the fact that some letters are more likely to be inserted than others). We assume that there is no cost for substituting a letter for itself (i.e., $\text{sub-cost}(x, x) = 0$).

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

Figure 2.18 Computation of minimum edit distance between *intention* and *execution* with the algorithm of Fig. 2.17, using Levenshtein distance with cost of 1 for insertions or deletions, 2 for substitutions.

While we worked our example with simple Levenshtein distance, the algorithm in Fig. 2.17 allows arbitrary weights on the operations. For spelling correction, for example, substitutions are more likely to happen between letters that are next to each other on the keyboard. The **Viterbi** algorithm is a probabilistic extension of minimum edit distance. Instead of computing the “minimum edit distance” between two strings, Viterbi computes the “maximum probability alignment” of one string with another. We’ll discuss this more in Chapter 8.

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ 1	↖↔↑ 2	↖↔↑ 3	↖↔↑ 4	↖↔↑ 5	↖↔↑ 6	↖↔↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖↔↑ 3	↖↔↑ 4	↖↔↑ 5	↖↔↑ 6	↖↔↑ 7	↖↔↑ 8	↑ 7	↖↔↑ 8	↖ 7
t	↑ 3	↖↔↑ 4	↖↔↑ 5	↖↔↑ 6	↖↔↑ 7	↖↔↑ 8	↖ 7	↔ 8	↖↔↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← 5	← 6	← 7	↔ 8	↖↔↑ 9	↖↔↑ 10	↑ 9
n	↑ 5	↑ 4	↖↔↑ 5	↖↔↑ 6	↖↔↑ 7	↖↔↑ 8	↖↔↑ 9	↖↔↑ 10	↖↔↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖↔↑ 6	↖↔↑ 7	↖↔↑ 8	↖↔↑ 9	↖ 8	← 9	← 10	↔ 11
i	↑ 7	↑ 6	↖↔↑ 7	↖↔↑ 8	↖↔↑ 9	↖↔↑ 10	↑ 9	↖ 8	← 9	← 10
o	↑ 8	↑ 7	↖↔↑ 8	↖↔↑ 9	↖↔↑ 10	↖↔↑ 11	↑ 10	↑ 9	↖ 8	← 9
n	↑ 9	↑ 8	↖↔↑ 9	↖↔↑ 10	↖↔↑ 11	↖↔↑ 12	↑ 11	↑ 10	↑ 9	↖ 8

Figure 2.19 When entering a value in each cell, we mark which of the three neighboring cells we came from with up to three arrows. After the table is full we compute an **alignment** (minimum edit path) by using a **backtrace**, starting at the **8** in the lower-right corner and following the arrows back. The sequence of bold cells represents one possible minimum cost alignment between the two strings. Diagram design after [Gusfield \(1997\)](#).

2.6 Summary

This chapter introduced a fundamental tool in language processing, the **regular expression**, and showed how to perform basic **text normalization** tasks including **word segmentation** and **normalization**, **sentence segmentation**, and **stemming**. We also introduce the important **minimum edit distance** algorithm for comparing strings. Here's a summary of the main points we covered about these ideas:

- The **regular expression** language is a powerful tool for pattern-matching.
- Basic operations in regular expressions include **concatenation** of symbols, **disjunction** of symbols ([], |, and .), **counters** (*, +, and {n,m}), **anchors** (^, \$) and precedence operators ((,)).
- **Word tokenization and normalization** are generally done by cascades of simple regular expressions substitutions or finite automata.
- The **Porter algorithm** is a simple and efficient way to do **stemming**, stripping off affixes. It does not have high accuracy but may be useful for some tasks.
- The **minimum edit distance** between two strings is the minimum number of operations it takes to edit one into the other. Minimum edit distance can be computed by **dynamic programming**, which also results in an **alignment** of the two strings.

Bibliographical and Historical Notes

Kleene (1951) and (1956) first defined regular expressions and the finite automaton, based on the McCulloch-Pitts neuron. Ken Thompson was one of the first to build regular expressions compilers into editors for text searching (Thompson, 1968). His editor *ed* included a command “g/regular expression/p”, or Global Regular Expression Print, which later became the Unix *grep* utility.

Text normalization algorithms have been applied since the beginning of the field. One of the earliest widely-used stemmers was Lovins (1968). Stemming was also applied early to the digital humanities, by Packard (1973), who built an affix-stripping morphological parser for Ancient Greek. Currently a wide vari-

ety of code for tokenization and normalization is available, such as the Stanford Tokenizer (<http://nlp.stanford.edu/software/tokenizer.shtml>) or specialized tokenizers for Twitter (O'Connor et al., 2010), or for sentiment (<http://sentiment.christopherpotts.net/tokenizing.html>). See Palmer (2012) for a survey of text preprocessing. NLTK is an essential tool that offers both useful Python libraries (<http://www.nltk.org>) and textbook descriptions (Bird et al., 2009) of many algorithms including text normalization and corpus interfaces.

For more on Herdan's law and Heaps' Law, see Herdan (1960, p. 28), Heaps (1978), Egghe (2007) and Baayen (2001); Yasseri et al. (2012) discuss the relationship with other measures of linguistic complexity. For more on edit distance, see the excellent Gusfield (1997). Our example measuring the edit distance from ‘intention’ to ‘execution’ was adapted from Kruskal (1983). There are various publicly available packages to compute edit distance, including Unix `diff` and the NIST `sclite` program (NIST, 2005).

In his autobiography Bellman (1984) explains how he originally came up with the term *dynamic programming*:

“...The 1950s were not good years for mathematical research. [the] Secretary of Defense ...had a pathological fear and hatred of the word, research... I decided therefore to use the word, “programming”. I wanted to get across the idea that this was dynamic, this was multi-stage... I thought, let’s ... take a word that has an absolutely precise meaning, namely dynamic... it’s impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It’s impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to.”

Exercises

- 2.1** Write regular expressions for the following languages.
 1. the set of all alphabetic strings;
 2. the set of all lower case alphabetic strings ending in a *b*;
 3. the set of all strings from the alphabet *a,b* such that each *a* is immediately preceded by and immediately followed by a *b*;
- 2.2** Write regular expressions for the following languages. By “word”, we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth.
 1. the set of all strings with two consecutive repeated words (e.g., “Humbert Humbert” and “the the” but not “the bug” or “the big bug”);
 2. all strings that start at the beginning of the line with an integer and that end at the end of the line with a word;
 3. all strings that have both the word *grotto* and the word *raven* in them (but not, e.g., words like *grottos* that merely *contain* the word *grotto*);
 4. write a pattern that places the first word of an English sentence in a register. Deal with punctuation.

- 2.3 Implement an ELIZA-like program, using substitutions such as those described on page 10. You might want to choose a different domain than a Rogerian psychologist, although keep in mind that you would need a domain in which your program can legitimately engage in a lot of simple repetition.
- 2.4 Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of “leda” to “deal”. Show your work (using the edit distance grid).
- 2.5 Figure out whether *drive* is closer to *brief* or to *divers* and what the edit distance is to each. You may use any version of *distance* that you like.
- 2.6 Now implement a minimum edit distance algorithm and use your hand-computed results to check your code.
- 2.7 Augment the minimum edit distance algorithm to output an alignment; you will need to store pointers and add a stage to compute the backtrace.

CHAPTER

3

N-gram Language Models

“You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Random sentence generated from a Jane Austen trigram model

Predicting is difficult—especially about the future, as the old quip goes. But how about predicting something that seems much easier, like the next few words someone is going to say? What word, for example, is likely to follow

Please turn your homework ...

Hopefully, most of you concluded that a very likely word is *in*, or possibly *over*, but probably not *refrigerator* or *the*. In the following sections we will formalize this intuition by introducing models that assign a **probability** to each possible next word. The same models will also serve to assign a probability to an entire sentence. Such a model, for example, could predict that the following sequence has a much higher probability of appearing in a text:

all of a sudden I notice three guys standing on the sidewalk

than does this same set of words in a different order:

on guys all I of notice sidewalk three a sudden standing the

Why would you want to predict upcoming words, or assign probabilities to sentences? Probabilities are essential in any task in which we have to identify words in noisy, ambiguous input, like **speech recognition**. For a speech recognizer to realize that you said *I will be back soonish* and not *I will be bassoon dish*, it helps to know that *back soonish* is a much more probable sequence than *bassoon dish*. For writing tools like **spelling correction** or **grammatical error correction**, we need to find and correct errors in writing like *Their are two midterms*, in which *There* was mistyped as *Their*, or *Everything has improve*, in which *improve* should have been *improved*. The phrase *There are* will be much more probable than *Their are*, and *has improved* than *has improve*, allowing us to help users by detecting and correcting these errors.

Assigning probabilities to sequences of words is also essential in **machine translation**. Suppose we are translating a Chinese source sentence:

他 向 记者 介绍了 主要 内容

He to reporters introduced main content

As part of the process we might have built the following set of potential rough English translations:

he introduced reporters to the main contents of the statement

he briefed to reporters the main contents of the statement

he briefed reporters on the main contents of the statement

A probabilistic model of word sequences could suggest that *briefed reporters on* is a more probable English phrase than *briefed to reporters* (which has an awkward *to* after *briefed*) or *introduced reporters to* (which uses a verb that is less fluent English in this context), allowing us to correctly select the boldfaced sentence above.

Probabilities are also important for **augmentative and alternative communication** systems (Trnka et al. 2007, Kane et al. 2017). People often use such AAC devices if they are physically unable or sign but can instead use eye gaze or other specific movements to select words from a menu to be spoken by the system. Word prediction can be used to suggest likely words for the menu.

Models that assign probabilities to sequences of words are called **language models** or **LMs**. In this chapter we introduce the simplest model that assigns probabilities to sentences and sequences of words, the **n-gram**. An n-gram is a sequence of N words: a 2-gram (or **bigram**) is a two-word sequence of words like “please turn”, “turn your”, or “your homework”, and a 3-gram (or **trigram**) is a three-word sequence of words like “please turn your”, or “turn your homework”. We’ll see how to use n-gram models to estimate the probability of the last word of an n-gram given the previous words, and also to assign probabilities to entire sequences. In a bit of terminological ambiguity, we usually drop the word “model”, and thus the term **n-gram** is used to mean either the word sequence itself or the predictive model that assigns it a probability. In later chapters we’ll introduce more sophisticated language models like the RNN LMs of Chapter 9.

3.1 N-Grams

Let’s begin with the task of computing $P(w|h)$, the probability of a word w given some history h . Suppose the history h is “*its water is so transparent that*” and we want to know the probability that the next word is *the*:

$$P(\text{the}|\text{its water is so transparent that}). \quad (3.1)$$

One way to estimate this probability is from relative frequency counts: take a very large corpus, count the number of times we see *its water is so transparent that*, and count the number of times this is followed by *the*. This would be answering the question “Out of the times we saw the history h , how many times was it followed by the word w ”, as follows:

$$\begin{aligned} P(\text{the}|\text{its water is so transparent that}) &= \\ \frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})} \end{aligned} \quad (3.2)$$

With a large enough corpus, such as the web, we can compute these counts and estimate the probability from Eq. 3.2. You should pause now, go to the web, and compute this estimate for yourself.

While this method of estimating probabilities directly from counts works fine in many cases, it turns out that even the web isn’t big enough to give us good estimates in most cases. This is because language is creative; new sentences are created all the time, and we won’t always be able to count entire sentences. Even simple extensions of the example sentence may have counts of zero on the web (such as “*Walden Pond’s water is so transparent that the*”; well, *used to* have counts of zero).

Similarly, if we wanted to know the joint probability of an entire sequence of words like *its water is so transparent*, we could do it by asking “out of all possible sequences of five words, how many of them are *its water is so transparent*?”. We would have to get the count of *its water is so transparent* and divide by the sum of the counts of all possible five word sequences. That seems rather a lot to estimate!

For this reason, we’ll need to introduce cleverer ways of estimating the probability of a word w given a history h , or the probability of an entire word sequence W . Let’s start with a little formalizing of notation. To represent the probability of a particular random variable X_i taking on the value “the”, or $P(X_i = \text{“the”})$, we will use the simplification $P(\text{the})$. We’ll represent a sequence of N words either as $w_1 \dots w_n$ or w_1^n (so the expression w_1^{n-1} means the string w_1, w_2, \dots, w_{n-1}). For the joint probability of each word in a sequence having a particular value $P(X = w_1, Y = w_2, Z = w_3, \dots, W = w_n)$ we’ll use $P(w_1, w_2, \dots, w_n)$.

Now how can we compute probabilities of entire sequences like $P(w_1, w_2, \dots, w_n)$? One thing we can do is decompose this probability using the **chain rule of probability**:

$$\begin{aligned} P(X_1 \dots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2) \dots P(X_n|X_1^{n-1}) \\ &= \prod_{k=1}^n P(X_k|X_1^{k-1}) \end{aligned} \tag{3.3}$$

Applying the chain rule to words, we get

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^n P(w_k|w_1^{k-1}) \end{aligned} \tag{3.4}$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words. Equation 3.4 suggests that we could estimate the joint probability of an entire sequence of words by multiplying together a number of conditional probabilities. But using the chain rule doesn’t really seem to help us! We don’t know any way to compute the exact probability of a word given a long sequence of preceding words, $P(w_n|w_1^{n-1})$. As we said above, we can’t just estimate by counting the number of times every word occurs following every long string, because language is creative and any particular context might have never occurred before!

The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can **approximate** the history by just the last few words.

bigram

The **bigram** model, for example, approximates the probability of a word given all the previous words $P(w_n|w_1^{n-1})$ by using only the conditional probability of the preceding word $P(w_n|w_{n-1})$. In other words, instead of computing the probability

$$P(\text{the}|\text{Walden Pond's water is so transparent that}) \tag{3.5}$$

we approximate it with the probability

$$P(\text{the}| \text{that}) \tag{3.6}$$

When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1}) \quad (3.7)$$

The assumption that the probability of a word depends only on the previous word is called a **Markov** assumption. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into the past) to the trigram (which looks two words into the past) and thus to the **n-gram** (which looks $n - 1$ words into the past).

Thus, the general equation for this n-gram approximation to the conditional probability of the next word in a sequence is

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \quad (3.8)$$

Given the bigram assumption for the probability of an individual word, we can compute the probability of a complete word sequence by substituting Eq. 3.7 into Eq. 3.4:

$$P(w_1^n) \approx \prod_{k=1}^n P(w_k|w_{k-1}) \quad (3.9)$$

How do we estimate these bigram or n-gram probabilities? An intuitive way to estimate probabilities is called **maximum likelihood estimation** or MLE. We get the MLE estimate for the parameters of an n-gram model by getting counts from a corpus, and **normalizing** the counts so that they lie between 0 and 1.¹

For example, to compute a particular bigram probability of a word y given a previous word x , we'll compute the count of the bigram $C(xy)$ and normalize by the sum of all the bigrams that share the same first word x :

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)} \quad (3.10)$$

We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1} (the reader should take a moment to be convinced of this):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (3.11)$$

Let's work through an example using a mini-corpus of three sentences. We'll first need to augment each sentence with a special symbol $\langle s \rangle$ at the beginning of the sentence, to give us the bigram context of the first word. We'll also need a special end-symbol. $\langle /s \rangle$ ²

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

¹ For probabilistic models, normalizing means dividing by some total count so that the resulting probabilities fall legally between 0 and 1.

² We need the end-symbol to make the bigram grammar a true probability distribution. Without an end-symbol, the sentence probabilities for all sentences of a given length would sum to one. This model would define an infinite set of probability distributions, with one distribution per sentence length. See Exercise 3.5.

Here are the calculations for some of the bigram probabilities from this corpus

$$\begin{aligned} P(\text{I}|\langle \text{s} \rangle) &= \frac{2}{3} = .67 & P(\text{Sam}|\langle \text{s} \rangle) &= \frac{1}{3} = .33 & P(\text{am}|\text{I}) &= \frac{2}{3} = .67 \\ P(\langle \text{/s} \rangle|\text{Sam}) &= \frac{1}{2} = 0.5 & P(\text{Sam}|\text{am}) &= \frac{1}{2} = .5 & P(\text{do}|\text{I}) &= \frac{1}{3} = .33 \end{aligned}$$

For the general case of MLE n-gram parameter estimation:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1} w_n)}{C(w_{n-N+1}^{n-1})} \quad (3.12)$$

relative frequency

Equation 3.12 (like Eq. 3.11) estimates the n-gram probability by dividing the observed frequency of a particular sequence by the observed frequency of a prefix. This ratio is called a **relative frequency**. We said above that this use of relative frequencies as a way to estimate probabilities is an example of maximum likelihood estimation or MLE. In MLE, the resulting parameter set maximizes the likelihood of the training set T given the model M (i.e., $P(T|M)$). For example, suppose the word *Chinese* occurs 400 times in a corpus of a million words like the Brown corpus. What is the probability that a random word selected from some other text of, say, a million words will be the word *Chinese*? The MLE of its probability is $\frac{400}{1000000}$ or .0004. Now .0004 is not the best possible estimate of the probability of *Chinese* occurring in all situations; it might turn out that in some other corpus or context *Chinese* is a very unlikely word. But it is the probability that makes it *most likely* that *Chinese* will occur 400 times in a million-word corpus. We present ways to modify the MLE estimates slightly to get better probability estimates in Section 3.4.

Let's move on to some examples from a slightly larger corpus than our 14-word example above. We'll use data from the now-defunct Berkeley Restaurant Project, a dialogue system from the last century that answered questions about a database of restaurants in Berkeley, California (Jurafsky et al., 1994). Here are some text-normalized sample user queries (a sample of 9332 sentences is on the website):

can you tell me about any good cantonese restaurants close by
mid priced thai food is what i'm looking for
tell me about chez panisse
can you give me a listing of the kinds of food that are available
i'm looking for a good place to eat breakfast
when is caffe venezia open during the day

Figure 3.1 shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of seven words would be even more sparse.

Figure 3.2 shows the bigram probabilities after normalization (dividing each cell in Fig. 3.1 by the appropriate unigram for its row, taken from the following set of unigram probabilities):

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Here are a few other useful probabilities:

$$\begin{aligned} P(\text{i}|\langle \text{s} \rangle) &= 0.25 & P(\text{english}|\text{want}) &= 0.0011 \\ P(\text{food}|\text{english}) &= 0.5 & P(\langle \text{/s} \rangle|\text{food}) &= 0.68 \end{aligned}$$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

$$\begin{aligned}
 P(<\text{s}> \text{ i want english food } </\text{s}>) \\
 &= P(\text{i}|\text{s}>)P(\text{want}|\text{i})P(\text{english}|\text{want}) \\
 &\quad P(\text{food}|\text{english})P(</\text{s}>|\text{food}) \\
 &= .25 \times .33 \times 0.0011 \times 0.5 \times 0.68 \\
 &= .000031
 \end{aligned}$$

We leave it as Exercise 3.2 to compute the probability of *i want chinese food*.

What kinds of linguistic phenomena are captured in these bigram statistics? Some of the bigram probabilities above encode some facts that we think of as strictly **syntactic** in nature, like the fact that what comes after *eat* is usually a noun or an adjective, or that what comes after *to* is usually a verb. Others might be a fact about the personal assistant task, like the high probability of sentences beginning with the words *I*. And some might even be cultural rather than linguistic, like the higher probability that people are looking for Chinese versus English food.

Some practical issues: Although for pedagogical purposes we have only described bigram models, in practice it's more common to use **trigram** models, which condition on the previous two words rather than the previous word, or **4-gram** or even **5-gram** models, when there is sufficient training data. Note that for these larger n-grams, we'll need to assume extra context for the contexts to the left and right of the sentence end. For example, to compute trigram probabilities at the very beginning of the sentence, we can use two pseudo-words for the first trigram (i.e., $P(\text{I}|\text{s}><\text{s}>)$).

trigram
4-gram
5-gram

log probabilities

We always represent and compute language model probabilities in log format as **log probabilities**. Since probabilities are (by definition) less than or equal to 1, the more probabilities we multiply together, the smaller the product becomes. Multiplying enough n-grams together would result in numerical underflow. By using log probabilities instead of raw probabilities, we get numbers that are not as small.

Adding in log space is equivalent to multiplying in linear space, so we combine log probabilities by adding them. The result of doing all computation and storage in log space is that we only need to convert back into probabilities if we need to report them at the end; then we can just take the exp of the logprob:

$$p_1 \times p_2 \times p_3 \times p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4) \quad (3.13)$$

3.2 Evaluating Language Models

extrinsic evaluation

The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called **extrinsic evaluation**. Extrinsic evaluation is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus, for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription.

intrinsic evaluation

Unfortunately, running big NLP systems end-to-end is often very expensive. Instead, it would be nice to have a metric that can be used to quickly evaluate potential improvements in a language model. An **intrinsic evaluation** metric is one that measures the quality of a model independent of any application.

training set

For an intrinsic evaluation of a language model we need a **test set**. As with many of the statistical models in our field, the probabilities of an n-gram model come from the corpus it is trained on, the **training set** or **training corpus**. We can then measure the quality of an n-gram model by its performance on some unseen data called the **test set** or test corpus. We will also sometimes call test sets and other datasets that are not in our training sets **held out** corpora because we hold them out from the training data.

test set
held out

So if we are given a corpus of text and want to compare two different n-gram models, we divide the data into training and test sets, train the parameters of both models on the training set, and then compare how well the two trained models fit the test set.

But what does it mean to “fit the test set”? The answer is simple: whichever model assigns a **higher probability** to the test set—meaning it more accurately predicts the test set—is a better model. Given two probabilistic models, the better model is the one that has a tighter fit to the test data or that better predicts the details of the test data, and hence will assign a higher probability to the test data.

Since our evaluation metric is based on test set probability, it’s important not to let the test sentences into the training set. Suppose we are trying to compute the probability of a particular “test” sentence. If our test sentence is part of the training corpus, we will mistakenly assign it an artificially high probability when it occurs in the test set. We call this situation **training on the test set**. Training on the test set introduces a bias that makes the probabilities all look too high, and causes huge inaccuracies in **perplexity**, the probability-based metric we introduce below.

development test

Sometimes we use a particular test set so often that we implicitly tune to its characteristics. We then need a fresh test set that is truly unseen. In such cases, we call the initial test set the **development** test set or, **devset**. How do we divide our data into training, development, and test sets? We want our test set to be as large as possible, since a small test set may be accidentally unrepresentative, but we also want as much training data as possible. At the minimum, we would want to pick

the smallest test set that gives us enough statistical power to measure a statistically significant difference between two potential models. In practice, we often just divide our data into 80% training, 10% development, and 10% test. Given a large corpus that we want to divide into training and test, test data can either be taken from some continuous sequence of text inside the corpus, or we can remove smaller “stripes” of text from randomly selected parts of our corpus and combine them into a test set.

3.2.1 Perplexity

In practice we don't use raw probability as our metric for evaluating language models, but a variant called **perplexity**. The **perplexity** (sometimes called *PP* for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set $W = w_1 w_2 \dots w_N$:

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned} \quad (3.14)$$

We can use the chain rule to expand the probability of W :

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \quad (3.15)$$

Thus, if we are computing the perplexity of W with a bigram language model, we get:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}} \quad (3.16)$$

Note that because of the inverse in Eq. 3.15, the higher the conditional probability of the word sequence, the lower the perplexity. Thus, minimizing perplexity is equivalent to maximizing the test set probability according to the language model. What we generally use for word sequence in Eq. 3.15 or Eq. 3.16 is the entire sequence of words in some test set. Since this sequence will cross many sentence boundaries, we need to include the begin- and end-sentence markers $\langle s \rangle$ and $\langle /s \rangle$ in the probability computation. We also need to include the end-of-sentence marker $\langle /s \rangle$ (but not the beginning-of-sentence marker $\langle s \rangle$) in the total count of word tokens N .

There is another way to think about perplexity: as the **weighted average branching factor** of a language. The branching factor of a language is the number of possible next words that can follow any word. Consider the task of recognizing the digits in English (zero, one, two,..., nine), given that (both in some training set and in some test set) each of the 10 digits occurs with equal probability $P = \frac{1}{10}$. The perplexity of this mini-language is in fact 10. To see that, imagine a test string of digits of length N , and assume that in the training set all the digits occurred with equal probability. By Eq. 3.15, the perplexity will be

$$\begin{aligned}
 \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\
 &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\
 &= \frac{1}{10}^{-1} \\
 &= 10
 \end{aligned} \tag{3.17}$$

But suppose that the number zero is really frequent and occurs far more often than other numbers. Let's say that 0 occur 91 times in the training set, and each of the other digits occurred 1 time each. Now we see the following test set: 0 0 0 0 3 0 0 0 0. We should expect the perplexity of this test set to be lower since most of the time the next number will be zero, which is very predictable, i.e. has a high probability. Thus, although the branching factor is still 10, the perplexity or *weighted* branching factor is smaller. We leave this exact calculation as exercise 12.

We see in Section 3.7 that perplexity is also closely related to the information-theoretic notion of entropy.

Finally, let's look at an example of how perplexity can be used to compare different n-gram models. We trained unigram, bigram, and trigram grammars on 38 million words (including start-of-sentence tokens) from the *Wall Street Journal*, using a 19,979 word vocabulary. We then computed the perplexity of each of these models on a test set of 1.5 million words with Eq. 3.16. The table below shows the perplexity of a 1.5 million word WSJ test set according to each of these grammars.

	Unigram	Bigram	Trigram
Perplexity	962	170	109

As we see above, the more information the n-gram gives us about the word sequence, the lower the perplexity (since as Eq. 3.15 showed, perplexity is related inversely to the likelihood of the test sequence according to the model).

Note that in computing perplexities, the n-gram model P must be constructed without any knowledge of the test set or any prior knowledge of the vocabulary of the test set. Any kind of knowledge of the test set can cause the perplexity to be artificially low. The perplexity of two language models is only comparable if they use identical vocabularies.

An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition or machine translation. Nonetheless, because perplexity often correlates with such improvements, it is commonly used as a quick check on an algorithm. But a model's improvement in perplexity should always be confirmed by an end-to-end evaluation of a real task before concluding the evaluation of the model.

3.3 Generalization and Zeros

The n-gram model, like many statistical models, is dependent on the training corpus. One implication of this is that the probabilities often encode specific facts about a given training corpus. Another implication is that n-grams do a better and better job of modeling the training corpus as we increase the value of N .

We can visualize both of these facts by borrowing the technique of [Shannon \(1951\)](#) and [Miller and Selfridge \(1950\)](#) of generating random sentences from different n-gram models. It's simplest to visualize how this works for the unigram case. Imagine all the words of the English language covering the probability space between 0 and 1, each word covering an interval proportional to its frequency. We choose a random value between 0 and 1 and print the word whose interval includes this chosen value. We continue choosing random numbers and generating words until we randomly generate the sentence-final token </s>. We can use the same technique to generate bigrams by first generating a random bigram that starts with <s> (according to its bigram probability). Let's say the second word of that bigram is w . We next chose a random bigram starting with w (again, drawn according to its bigram probability), and so on.

To give an intuition for the increasing power of higher-order n-grams, Fig. 3.3 shows random sentences generated from unigram, bigram, trigram, and 4-gram models trained on Shakespeare's works.

1 gram	<ul style="list-style-type: none"> –To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	<ul style="list-style-type: none"> –Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	<ul style="list-style-type: none"> –Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	<ul style="list-style-type: none"> –King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

Figure 3.3 Eight sentences randomly generated from four n-grams computed from Shakespeare's works. All characters were mapped to lower-case and punctuation marks were treated as words. Output is hand-corrected for capitalization to improve readability.

The longer the context on which we train the model, the more coherent the sentences. In the unigram sentences, there is no coherent relation between words or any sentence-final punctuation. The bigram sentences have some local word-to-word coherence (especially if we consider that punctuation counts as a word). The trigram and 4-gram sentences are beginning to look a lot like Shakespeare. Indeed, a careful investigation of the 4-gram sentences shows that they look a little too much like Shakespeare. The words *It cannot be but so* are directly from *King John*. This is because, not to put the knock on Shakespeare, his oeuvre is not very large as corpora go ($N = 884,647, V = 29,066$), and our n-gram probability matrices are ridiculously sparse. There are $V^2 = 844,000,000$ possible bigrams alone, and the number of possible 4-grams is $V^4 = 7 \times 10^{17}$. Thus, once the generator has chosen the first 4-gram (*It cannot be but*), there are only five possible continuations (*that, I, he, thou, and so*); indeed, for many 4-grams, there is only one continuation.

To get an idea of the dependence of a grammar on its training set, let's look at an n-gram grammar trained on a completely different corpus: the *Wall Street Journal* (WSJ) newspaper. Shakespeare and the *Wall Street Journal* are both English, so we might expect some overlap between our n-grams for the two genres. Fig. 3.4

shows sentences generated by unigram, bigram, and trigram grammars trained on 40 million words from WSJ.

1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 3.4 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

Compare these examples to the pseudo-Shakespeare in Fig. 3.3. While they both model “English-like sentences”, there is clearly no overlap in generated sentences, and little overlap even in small phrases. Statistical models are likely to be pretty useless as predictors if the training sets and the test sets are as different as Shakespeare and WSJ.

How should we deal with this problem when we build n-gram models? One step is to be sure to use a training corpus that has a similar **genre** to whatever task we are trying to accomplish. To build a language model for translating legal documents, we need a training corpus of legal documents. To build a language model for a question-answering system, we need a training corpus of questions.

It is equally important to get training data in the appropriate **dialect**, especially when processing social media posts or spoken transcripts. Thus tweets in AAVE (African American Vernacular English) often use words like *finna*—an auxiliary verb that marks immediate future tense—that don’t occur in other dialects, or spellings like *den* for *then*, in tweets like this one (Blodgett and O’Connor, 2017):

(3.18) Bored af den my phone finna die!!!

while tweets from varieties like Nigerian English have markedly different vocabulary and n-gram patterns from American English (Jurgens et al., 2017):

(3.19) @username R u a wizard or wat gan sef: in d mornin - u tweet, afternoon - u
tweet, nyt gan u dey tweet. beta get ur IT placement wiv twitter

Matching genres and dialects is still not sufficient. Our models may still be subject to the problem of **sparsity**. For any n-gram that occurred a sufficient number of times, we might have a good estimate of its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it. That is, we’ll have many cases of putative “zero probability n-grams” that should really have some non-zero probability. Consider the words that follow the bigram *denied the* in the WSJ Treebank3 corpus, together with their counts:

denied the allegations:	5
denied the speculation:	2
denied the rumors:	1
denied the report:	1

But suppose our test set has phrases like:

denied the offer
denied the loan

Our model will incorrectly estimate that the $P(\text{offer}|\text{denied the})$ is 0!

zeros

These **zeros**—things that don't ever occur in the training set but do occur in the test set—are a problem for two reasons. First, their presence means we are underestimating the probability of all sorts of words that might occur, which will hurt the performance of any application we want to run on this data.

Second, if the probability of any word in the test set is 0, the entire probability of the test set is 0. By definition, perplexity is based on the inverse probability of the test set. Thus if some words have zero probability, we can't compute perplexity at all, since we can't divide by 0!

3.3.1 Unknown Words

The previous section discussed the problem of words whose bigram probability is zero. But what about words we simply have never seen before?

closed vocabulary

Sometimes we have a language task in which this can't happen because we know all the words that can occur. In such a **closed vocabulary** system the test set can only contain words from this lexicon, and there will be no unknown words. This is a reasonable assumption in some domains, such as speech recognition or machine translation, where we have a pronunciation dictionary or a phrase table that are fixed in advance, and so the language model can only use the words in that dictionary or phrase table.

OOV open vocabulary

In other cases we have to deal with words we haven't seen before, which we'll call **unknown** words, or **out of vocabulary (OOV)** words. The percentage of OOV words that appear in the test set is called the **OOV rate**. An **open vocabulary** system is one in which we model these potential unknown words in the test set by adding a pseudo-word called <UNK>.

There are two common ways to train the probabilities of the unknown word model <UNK>. The first one is to turn the problem back into a closed vocabulary one by choosing a fixed vocabulary in advance:

1. **Choose a vocabulary** (word list) that is fixed in advance.
2. **Convert** in the training set any word that is not in this set (any OOV word) to the unknown word token <UNK> in a text normalization step.
3. **Estimate** the probabilities for <UNK> from its counts just like any other regular word in the training set.

The second alternative, in situations where we don't have a prior vocabulary in advance, is to create such a vocabulary implicitly, replacing words in the training data by <UNK> based on their frequency. For example we can replace by <UNK> all words that occur fewer than n times in the training set, where n is some small number, or equivalently select a vocabulary size V in advance (say 50,000) and choose the top V words by frequency and replace the rest by UNK. In either case we then proceed to train the language model as before, treating <UNK> like a regular word.

The exact choice of <UNK> model does have an effect on metrics like perplexity. A language model can achieve low perplexity by choosing a small vocabulary and assigning the unknown word a high probability. For this reason, perplexities should only be compared across language models with the same vocabularies (Buck et al., 2014).

3.4 Smoothing

smoothing
discounting

What do we do with words that are in our vocabulary (they are not unknown words) but appear in a test set in an unseen context (for example they appear after a word they never appeared after in training)? To keep a language model from assigning zero probability to these unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen. This modification is called **smoothing** or **discounting**. In this section and the following ones we'll introduce a variety of ways to do smoothing: **add-1 smoothing**, **add-k smoothing**, **stupid backoff**, and **Kneser-Ney smoothing**.

Laplace smoothing

3.4.1 Laplace Smoothing

The simplest way to do smoothing is to add one to all the bigram counts, before we normalize them into probabilities. All the counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on. This algorithm is called **Laplace smoothing**. Laplace smoothing does not perform well enough to be used in modern n-gram models, but it usefully introduces many of the concepts that we see in other smoothing algorithms, gives a useful baseline, and is also a practical smoothing algorithm for other tasks like **text classification** (Chapter 4).

Let's start with the application of Laplace smoothing to unigram probabilities. Recall that the unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N :

$$P(w_i) = \frac{c_i}{N}$$

add-one

Laplace smoothing merely adds one to each count (hence its alternate name **add-one smoothing**). Since there are V words in the vocabulary and each one was incremented, we also need to adjust the denominator to take into account the extra V observations. (What happens to our P values if we don't increase the denominator?)

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V} \tag{3.20}$$

Instead of changing both the numerator and denominator, it is convenient to describe how a smoothing algorithm affects the numerator, by defining an **adjusted count** c^* . This adjusted count is easier to compare directly with the MLE counts and can be turned into a probability like an MLE count by normalizing by N . To define this count, since we are only changing the numerator in addition to adding 1 we'll also need to multiply by a normalization factor $\frac{N}{N+V}$:

$$c_i^* = (c_i + 1) \frac{N}{N + V} \tag{3.21}$$

discounting
discount

We can now turn c_i^* into a probability P_i^* by normalizing by N .

A related way to view smoothing is as **discounting** (lowering) some non-zero counts in order to get the probability mass that will be assigned to the zero counts. Thus, instead of referring to the discounted counts c^* , we might describe a smoothing algorithm in terms of a relative **discount** d_c , the ratio of the discounted counts to the original counts:

$$d_c = \frac{c^*}{c}$$

Now that we have the intuition for the unigram case, let's smooth our Berkeley Restaurant Project bigrams. Figure 3.5 shows the add-one smoothed counts for the bigrams in Fig. 3.1.

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Figure 3.5 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

Figure 3.6 shows the add-one smoothed probabilities for the bigrams in Fig. 3.2. Recall that normal bigram probabilities are computed by normalizing each row of counts by the unigram count:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \quad (3.22)$$

For add-one smoothed bigram counts, we need to augment the unigram count by the number of total word types in the vocabulary V :

$$P_{\text{Laplace}}^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{\sum_w (C(w_{n-1}w) + 1)} = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V} \quad (3.23)$$

Thus, each of the unigram counts given in the previous section will need to be augmented by $V = 1446$. The result is the smoothed bigram probabilities in Fig. 3.6.

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Figure 3.6 Add-one smoothed bigram probabilities for eight of the words (out of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero probabilities are in gray.

It is often convenient to reconstruct the count matrix so we can see how much a smoothing algorithm has changed the original counts. These adjusted counts can be computed by Eq. 3.24. Figure 3.7 shows the reconstructed counts.

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V} \quad (3.24)$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figure 3.7 Add-one reconstituted counts for eight words (of $V = 1446$) in the BeRP corpus of 9332 sentences. Previously-zero counts are in gray.

Note that add-one smoothing has made a very big change to the counts. $C(want\ to)$ changed from 609 to 238! We can see this in probability space as well: $P(to|want)$ decreases from .66 in the unsmoothed case to .26 in the smoothed case. Looking at the discount d (the ratio between new and old counts) shows us how strikingly the counts for each prefix word have been reduced; the discount for the bigram *want to* is .39, while the discount for *Chinese food* is .10, a factor of 10!

The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros.

3.4.2 Add-k smoothing

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k (.5? .05? .01?). This algorithm is therefore called **add-k smoothing**.

$$P_{\text{Add-k}}^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV} \quad (3.25)$$

Add-k smoothing requires that we have a method for choosing k ; this can be done, for example, by optimizing on a **devset**. Although add-k is useful for some tasks (including text classification), it turns out that it still doesn't work well for language modeling, generating counts with poor variances and often inappropriate discounts (Gale and Church, 1994).

3.4.3 Backoff and Interpolation

The discounting we have been discussing so far can help solve the problem of zero frequency n-grams. But there is an additional source of knowledge we can draw on. If we are trying to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.

backoff

In other words, sometimes using **less context** is a good thing, helping to generalize more for contexts that the model hasn't learned much about. There are two ways to use this n-gram "hierarchy". In **backoff**, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram. By contrast, in **interpolation**, we always mix the probability estimates from all the n-gram estimators, weighing and combining the trigram, bigram, and unigram counts.

interpolation

In simple linear interpolation, we combine different order n-grams by linearly interpolating all the models. Thus, we estimate the trigram probability $P(w_n|w_{n-2}w_{n-1})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by a λ :

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &\quad + \lambda_2 P(w_n|w_{n-1}) \\ &\quad + \lambda_3 P(w_n)\end{aligned}\tag{3.26}$$

such that the λ s sum to 1:

$$\sum_i \lambda_i = 1\tag{3.27}$$

In a slightly more sophisticated version of linear interpolation, each λ weight is computed by conditioning on the context. This way, if we have particularly accurate counts for a particular bigram, we assume that the counts of the trigrams based on this bigram will be more trustworthy, so we can make the λ s for those trigrams higher and thus give that trigram more weight in the interpolation. Equation 3.28 shows the equation for interpolation with context-conditioned weights:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &\quad + \lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &\quad + \lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}\tag{3.28}$$

How are these λ values set? Both the simple interpolation and conditional interpolation λ s are learned from a **held-out** corpus. A held-out corpus is an additional training corpus that we use to set hyperparameters like these λ values, by choosing the λ values that maximize the likelihood of the held-out corpus. That is, we fix the n-gram probabilities and then search for the λ values that—when plugged into Eq. 3.26—give us the highest probability of the held-out set. There are various ways to find this optimal set of λ s. One way is to use the **EM** algorithm, an iterative learning algorithm that converges on locally optimal λ s (Jelinek and Mercer, 1980).

In a **backoff** n-gram model, if the n-gram we need has zero counts, we approximate it by backing off to the (N-1)-gram. We continue backing off until we reach a history that has some counts.

In order for a backoff model to give a correct probability distribution, we have to **discount** the higher-order n-grams to save some probability mass for the lower order n-grams. Just as with add-one smoothing, if the higher-order n-grams aren't discounted and we just used the undiscounted MLE probability, then as soon as we replaced an n-gram which has zero probability with a lower-order n-gram, we would be adding probability mass, and the total probability assigned to all possible strings by the language model would be greater than 1! In addition to this explicit discount factor, we'll need a function α to distribute this probability mass to the lower order n-grams.

This kind of backoff with discounting is also called **Katz backoff**. In Katz back-off we rely on a discounted probability P^* if we've seen this n-gram before (i.e., if we have non-zero counts). Otherwise, we recursively back off to the Katz probability for the shorter-history (N-1)-gram. The probability for a backoff n-gram P_{BO} is

thus computed as follows:

$$P_{\text{BO}}(w_n | w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n | w_{n-N+1}^{n-1}), & \text{if } C(w_{n-N+1}^n) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{\text{BO}}(w_n | w_{n-N+2}^{n-1}), & \text{otherwise.} \end{cases} \quad (3.29)$$

Good-Turing

Katz backoff is often combined with a smoothing method called **Good-Turing**. The combined **Good-Turing backoff** algorithm involves quite detailed computation for estimating the Good-Turing smoothing and the P^* and α values.

3.5 Kneser-Ney Smoothing

Kneser-Ney

One of the most commonly used and best performing n-gram smoothing methods is the interpolated **Kneser-Ney** algorithm ([Kneser and Ney 1995](#), [Chen and Goodman 1998](#)).

Kneser-Ney has its roots in a method called **absolute discounting**. Recall that **discounting** of the counts for frequent n-grams is necessary to save some probability mass for the smoothing algorithm to distribute to the unseen n-grams.

To see this, we can use a clever idea from [Church and Gale \(1991\)](#). Consider an n-gram that has count 4. We need to discount this count by some amount. But how much should we discount it? Church and Gale's clever idea was to look at a held-out corpus and just see what the count is for all those bigrams that had count 4 in the training set. They computed a bigram grammar from 22 million words of AP newswire and then checked the counts of each of these bigrams in another 22 million words. On average, a bigram that occurred 4 times in the first 22 million words occurred 3.23 times in the next 22 million words. The following table from [Church and Gale \(1991\)](#) shows these counts for bigrams with c from 0 to 9:

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Figure 3.8 For all bigrams in 22 million words of AP newswire of count 0, 1, 2,...,9, the counts of these bigrams in a held-out corpus also of 22 million words.

Absolute discounting

The astute reader may have noticed that except for the held-out counts for 0 and 1, all the other bigram counts in the held-out set could be estimated pretty well by just subtracting 0.75 from the count in the training set! **Absolute discounting** formalizes this intuition by subtracting a fixed (absolute) discount d from each count. The intuition is that since we have good estimates already for the very high counts, a small discount d won't affect them much. It will mainly modify the smaller counts,

for which we don't necessarily trust the estimate anyway, and Fig. 3.8 suggests that in practice this discount is actually a good one for bigrams with counts 2 through 9. The equation for interpolated absolute discounting applied to bigrams:

$$P_{\text{AbsoluteDiscounting}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_v C(w_{i-1}v)} + \lambda(w_{i-1})P(w_i) \quad (3.30)$$

The first term is the discounted bigram, and the second term is the unigram with an interpolation weight λ . We could just set all the d values to .75, or we could keep a separate discount value of 0.5 for the bigrams with counts of 1.

Kneser-Ney discounting (Kneser and Ney, 1995) augments absolute discounting with a more sophisticated way to handle the lower-order unigram distribution. Consider the job of predicting the next word in this sentence, assuming we are interpolating a bigram and a unigram model.

I can't see without my reading _____.

The word *glasses* seems much more likely to follow here than, say, the word *Kong*, so we'd like our unigram model to prefer *glasses*. But in fact it's *Kong* that is more common, since *Hong Kong* is a very frequent word. A standard unigram model will assign *Kong* a higher probability than *glasses*. We would like to capture the intuition that although *Kong* is frequent, it is mainly only frequent in the phrase *Hong Kong*, that is, after the word *Hong*. The word *glasses* has a much wider distribution.

In other words, instead of $P(w)$, which answers the question "How likely is w ? ", we'd like to create a unigram model that we might call $P_{\text{CONTINUATION}}$, which answers the question "How likely is w to appear as a novel continuation? ". How can we estimate this probability of seeing the word w as a novel continuation, in a new unseen context? The Kneser-Ney intuition is to base our estimate of $P_{\text{CONTINUATION}}$ on the *number of different contexts word w has appeared in*, that is, the number of bigram types it completes. Every bigram type was a novel continuation the first time it was seen. We hypothesize that words that have appeared in more contexts in the past are more likely to appear in some new context as well. The number of times a word w appears as a novel continuation can be expressed as:

$$P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}| \quad (3.31)$$

To turn this count into a probability, we normalize by the total number of word bigram types. In summary:

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|} \quad (3.32)$$

An equivalent formulation based on a different metaphor is to use the number of word types seen to precede w (Eq. 3.31 repeated):

$$P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}| \quad (3.33)$$

normalized by the number of words preceding all words, as follows:

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(vw') > 0\}|} \quad (3.34)$$

A frequent word (*Kong*) occurring in only one context (*Hong*) will have a low continuation probability.

Interpolated Kneser-Ney

The final equation for **Interpolated Kneser-Ney** smoothing for bigrams is then:

$$P_{\text{KN}}(w_i|w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{\text{CONTINUATION}}(w_i) \quad (3.35)$$

The λ is a normalizing constant that is used to distribute the probability mass we've discounted.:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}| \quad (3.36)$$

The first term, $\frac{d}{\sum_v C(w_{i-1}v)}$, is the normalized discount. The second term, $|\{w : C(w_{i-1}w) > 0\}|$, is the number of word types that can follow w_{i-1} or, equivalently, the number of word types that we discounted; in other words, the number of times we applied the normalized discount.

The general recursive formulation is as follows:

$$P_{\text{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max(c_{\text{KN}}(w_{i-n+1}^i) - d, 0)}{\sum_v c_{\text{KN}}(w_{i-n+1}^i v)} + \lambda(w_{i-n+1}^{i-1})P_{\text{KN}}(w_i|w_{i-n+2}^{i-1}) \quad (3.37)$$

where the definition of the count c_{KN} depends on whether we are counting the highest-order n-gram being interpolated (for example trigram if we are interpolating trigram, bigram, and unigram) or one of the lower-order n-grams (bigram or unigram if we are interpolating trigram, bigram, and unigram):

$$c_{\text{KN}}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{for the highest order} \\ \text{continuationcount}(\cdot) & \text{for lower orders} \end{cases} \quad (3.38)$$

The continuation count is the number of unique single word contexts for \cdot .

At the termination of the recursion, unigrams are interpolated with the uniform distribution, where the parameter ϵ is the empty string:

$$P_{\text{KN}}(w) = \frac{\max(c_{\text{KN}}(w) - d, 0)}{\sum_{w'} c_{\text{KN}}(w')} + \lambda(\epsilon) \frac{1}{V} \quad (3.39)$$

If we want to include an unknown word `<UNK>`, it's just included as a regular vocabulary entry with count zero, and hence its probability will be a lambda-weighted uniform distribution $\frac{\lambda(\epsilon)}{V}$.

modified Kneser-Ney

The best-performing version of Kneser-Ney smoothing is called **modified Kneser-Ney** smoothing, and is due to [Chen and Goodman \(1998\)](#). Rather than use a single fixed discount d , modified Kneser-Ney uses three different discounts d_1 , d_2 , and d_{3+} for n-grams with counts of 1, 2 and three or more, respectively. See [Chen and Goodman \(1998, p. 19\)](#) or [Heafield et al. \(2013\)](#) for the details.

3.6 The Web and Stupid Backoff

By using text from the web, it is possible to build extremely large language models. In 2006 Google released a very large set of n-gram counts, including n-grams (1-grams through 5-grams) from all the five-word sequences that appear at least

40 times from 1,024,908,267,229 words of running text on the web; this includes 1,176,470,663 five-word sequences using over 13 million unique words types (Franz and Brants, 2006). Some examples:

4-gram	Count
serve as the incoming	92
serve as the incubator	99
serve as the independent	794
serve as the index	223
serve as the indication	72
serve as the indicator	120
serve as the indicators	45
serve as the indispensable	111
serve as the indispensable	40
serve as the individual	234

Efficiency considerations are important when building language models that use such large sets of n-grams. Rather than store each word as a string, it is generally represented in memory as a 64-bit hash number, with the words themselves stored on disk. Probabilities are generally quantized using only 4-8 bits (instead of 8-byte floats), and n-grams are stored in reverse tries.

Bloom filters

N-grams can also be shrunk by pruning, for example only storing n-grams with counts greater than some threshold (such as the count threshold of 40 used for the Google n-gram release) or using entropy to prune less-important n-grams (Stolcke, 1998). Another option is to build approximate language models using techniques like **Bloom filters** (Talbot and Osborne 2007, Church et al. 2007). Finally, efficient language model toolkits like KenLM (Heafield 2011, Heafield et al. 2013) use sorted arrays, efficiently combine probabilities and backoffs in a single value, and use merge sorts to efficiently build the probability tables in a minimal number of passes through a large corpus.

stupid backoff

Although with these toolkits it is possible to build web-scale language models using full Kneser-Ney smoothing, Brants et al. (2007) show that with very large language models a much simpler algorithm may be sufficient. The algorithm is called **stupid backoff**. Stupid backoff gives up the idea of trying to make the language model a true probability distribution. There is no discounting of the higher-order probabilities. If a higher-order n-gram has a zero count, we simply backoff to a lower order n-gram, weighed by a fixed (context-independent) weight. This algorithm does not produce a probability distribution, so we'll follow Brants et al. (2007) in referring to it as S :

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ \lambda S(w_i|w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases} \quad (3.40)$$

The backoff terminates in the unigram, which has probability $S(w) = \frac{\text{count}(w)}{N}$. Brants et al. (2007) find that a value of 0.4 worked well for λ .

3.7 Advanced: Perplexity's Relation to Entropy

We introduced perplexity in Section 3.2.1 as a way to evaluate n-gram models on a test set. A better n-gram model is one that assigns a higher probability to the

Entropy

test data, and perplexity is a normalized version of the probability of the test set. The perplexity measure actually arises from the information-theoretic concept of cross-entropy, which explains otherwise mysterious properties of perplexity (why the inverse probability, for example?) and its relationship to entropy. **Entropy** is a measure of information. Given a random variable X ranging over whatever we are predicting (words, letters, parts of speech, the set of which we'll call χ) and with a particular probability function, call it $p(x)$, the entropy of the random variable X is:

$$H(X) = - \sum_{x \in \chi} p(x) \log_2 p(x) \quad (3.41)$$

The log can, in principle, be computed in any base. If we use log base 2, the resulting value of entropy will be measured in **bits**.

One intuitive way to think about entropy is as a lower bound on the number of bits it would take to encode a certain decision or piece of information in the optimal coding scheme.

Consider an example from the standard information theory textbook [Cover and Thomas \(1991\)](#). Imagine that we want to place a bet on a horse race but it is too far to go all the way to Yonkers Racetrack, so we'd like to send a short message to the bookie to tell him which of the eight horses to bet on. One way to encode this message is just to use the binary representation of the horse's number as the code; thus, horse 1 would be **001**, horse 2 **010**, horse 3 **011**, and so on, with horse 8 coded as **000**. If we spend the whole day betting and each horse is coded with 3 bits, on average we would be sending 3 bits per race.

Can we do better? Suppose that the spread is the actual distribution of the bets placed and that we represent it as the prior probability of each horse as follows:

Horse 1	$\frac{1}{2}$	Horse 5	$\frac{1}{64}$
Horse 2	$\frac{1}{4}$	Horse 6	$\frac{1}{64}$
Horse 3	$\frac{1}{8}$	Horse 7	$\frac{1}{64}$
Horse 4	$\frac{1}{16}$	Horse 8	$\frac{1}{64}$

The entropy of the random variable X that ranges over horses gives us a lower bound on the number of bits and is

$$\begin{aligned} H(X) &= - \sum_{i=1}^{i=8} p(i) \log p(i) \\ &= -\frac{1}{2} \log \frac{1}{2} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{8} \log \frac{1}{8} - \frac{1}{16} \log \frac{1}{16} - 4(\frac{1}{64} \log \frac{1}{64}) \\ &= 2 \text{ bits} \end{aligned} \quad (3.42)$$

A code that averages 2 bits per race can be built with short encodings for more probable horses, and longer encodings for less probable horses. For example, we could encode the most likely horse with the code **0**, and the remaining horses as **10**, then **110**, **1110**, **111100**, **111101**, **111110**, and **111111**.

What if the horses are equally likely? We saw above that if we used an equal-length binary code for the horse numbers, each horse took 3 bits to code, so the average was 3. Is the entropy the same? In this case each horse would have a probability of $\frac{1}{8}$. The entropy of the choice of horses is then

$$H(X) = - \sum_{i=1}^{i=8} \frac{1}{8} \log \frac{1}{8} = -\log \frac{1}{8} = 3 \text{ bits} \quad (3.43)$$

Until now we have been computing the entropy of a single variable. But most of what we will use entropy for involves *sequences*. For a grammar, for example, we will be computing the entropy of some sequence of words $W = \{w_0, w_1, w_2, \dots, w_n\}$. One way to do this is to have a variable that ranges over sequences of words. For example we can compute the entropy of a random variable that ranges over all finite sequences of words of length n in some language L as follows:

$$H(w_1, w_2, \dots, w_n) = - \sum_{W_1^n \in L} p(W_1^n) \log p(W_1^n) \quad (3.44)$$

entropy rate

We could define the **entropy rate** (we could also think of this as the **per-word entropy**) as the entropy of this sequence divided by the number of words:

$$\frac{1}{n} H(W_1^n) = - \frac{1}{n} \sum_{W_1^n \in L} p(W_1^n) \log p(W_1^n) \quad (3.45)$$

But to measure the true entropy of a language, we need to consider sequences of infinite length. If we think of a language as a stochastic process L that produces a sequence of words, and allow W to represent the sequence of words w_1, \dots, w_n , then L 's entropy rate $H(L)$ is defined as

$$\begin{aligned} H(L) &= \lim_{n \rightarrow \infty} \frac{1}{n} H(w_1, w_2, \dots, w_n) \\ &= - \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{W \in L} p(w_1, \dots, w_n) \log p(w_1, \dots, w_n) \end{aligned} \quad (3.46)$$

The Shannon-McMillan-Breiman theorem ([Algoet and Cover 1988](#), [Cover and Thomas 1991](#)) states that if the language is regular in certain ways (to be exact, if it is both stationary and ergodic),

$$H(L) = \lim_{n \rightarrow \infty} -\frac{1}{n} \log p(w_1 w_2 \dots w_n) \quad (3.47)$$

That is, we can take a single sequence that is long enough instead of summing over all possible sequences. The intuition of the Shannon-McMillan-Breiman theorem is that a long-enough sequence of words will contain in it many other shorter sequences and that each of these shorter sequences will reoccur in the longer sequence according to their probabilities.

Stationary

A stochastic process is said to be **stationary** if the probabilities it assigns to a sequence are invariant with respect to shifts in the time index. In other words, the probability distribution for words at time t is the same as the probability distribution at time $t + 1$. Markov models, and hence n-grams, are stationary. For example, in a bigram, P_i is dependent only on P_{i-1} . So if we shift our time index by x , P_{i+x} is still dependent on P_{i+x-1} . But natural language is not stationary, since as we show in Chapter 12, the probability of upcoming words can be dependent on events that were arbitrarily distant and time dependent. Thus, our statistical models only give an approximation to the correct distributions and entropies of natural language.

To summarize, by making some incorrect but convenient simplifying assumptions, we can compute the entropy of some stochastic process by taking a very long sample of the output and computing its average log probability.

cross-entropy

Now we are ready to introduce **cross-entropy**. The cross-entropy is useful when we don't know the actual probability distribution p that generated some data. It

allows us to use some m , which is a model of p (i.e., an approximation to p). The cross-entropy of m on p is defined by

$$H(p, m) = \lim_{n \rightarrow \infty} -\frac{1}{n} \sum_{W \in L} p(w_1, \dots, w_n) \log m(w_1, \dots, w_n) \quad (3.48)$$

That is, we draw sequences according to the probability distribution p , but sum the log of their probabilities according to m .

Again, following the Shannon-McMillan-Breiman theorem, for a stationary ergodic process:

$$H(p, m) = \lim_{n \rightarrow \infty} -\frac{1}{n} \log m(w_1 w_2 \dots w_n) \quad (3.49)$$

This means that, as for entropy, we can estimate the cross-entropy of a model m on some distribution p by taking a single sequence that is long enough instead of summing over all possible sequences.

What makes the cross-entropy useful is that the cross-entropy $H(p, m)$ is an upper bound on the entropy $H(p)$. For any model m :

$$H(p) \leq H(p, m) \quad (3.50)$$

This means that we can use some simplified model m to help estimate the true entropy of a sequence of symbols drawn according to probability p . The more accurate m is, the closer the cross-entropy $H(p, m)$ will be to the true entropy $H(p)$. Thus, the difference between $H(p, m)$ and $H(p)$ is a measure of how accurate a model is. Between two models m_1 and m_2 , the more accurate model will be the one with the lower cross-entropy. (The cross-entropy can never be lower than the true entropy, so a model cannot err by underestimating the true entropy.)

We are finally ready to see the relation between perplexity and cross-entropy as we saw it in Eq. 3.49. Cross-entropy is defined in the limit, as the length of the observed word sequence goes to infinity. We will need an approximation to cross-entropy, relying on a (sufficiently long) sequence of fixed length. This approximation to the cross-entropy of a model $M = P(w_i | w_{i-N+1} \dots w_{i-1})$ on a sequence of words W is

$$H(W) = -\frac{1}{N} \log P(w_1 w_2 \dots w_N) \quad (3.51)$$

perplexity The **perplexity** of a model P on a sequence of words W is now formally defined as the exp of this cross-entropy:

$$\begin{aligned} \text{Perplexity}(W) &= 2^{H(W)} \\ &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}} \end{aligned} \quad (3.52)$$

3.8 Summary

This chapter introduced language modeling and the n-gram, one of the most widely used tools in language processing.

- Language models offer a way to assign a probability to a sentence or other sequence of words, and to predict a word from preceding words.
- n-grams are Markov models that estimate words from a fixed window of previous words. n-gram probabilities can be estimated by counting in a corpus and normalizing (the **maximum likelihood estimate**).
- n-gram **language models** are evaluated extrinsically in some task, or intrinsically using **perplexity**.
- The **perplexity** of a test set according to a language model is the geometric mean of the inverse test set probability computed by the model.
- **Smoothing** algorithms provide a more sophisticated way to estimate the probability of n-grams. Commonly used smoothing algorithms for n-grams rely on lower-order n-gram counts through **backoff** or **interpolation**.
- Both backoff and interpolation require **discounting** to create a probability distribution.
- **Kneser-Ney** smoothing makes use of the probability of a word being a novel **continuation**. The interpolated **Kneser-Ney** smoothing algorithm mixes a discounted probability with a lower-order continuation probability.

Bibliographical and Historical Notes

The underlying mathematics of the n-gram was first proposed by [Markov \(1913\)](#), who used what are now called **Markov chains** (bigrams and trigrams) to predict whether an upcoming letter in Pushkin’s *Eugene Onegin* would be a vowel or a consonant. Markov classified 20,000 letters as V or C and computed the bigram and trigram probability that a given letter would be a vowel given the previous one or two letters. [Shannon \(1948\)](#) applied n-grams to compute approximations to English word sequences. Based on Shannon’s work, Markov models were commonly used in engineering, linguistic, and psychological work on modeling word sequences by the 1950s. In a series of extremely influential papers starting with [Chomsky \(1956\)](#) and including [Chomsky \(1957\)](#) and [Miller and Chomsky \(1963\)](#), Noam Chomsky argued that “finite-state Markov processes”, while a possibly useful engineering heuristic, were incapable of being a complete cognitive model of human grammatical knowledge. These arguments led many linguists and computational linguists to ignore work in statistical modeling for decades.

The resurgence of n-gram models came from Jelinek and colleagues at the IBM Thomas J. Watson Research Center, who were influenced by Shannon, and Baker at CMU, who was influenced by the work of Baum and colleagues. Independently these two labs successfully used n-grams in their speech recognition systems ([Baker 1975b](#), [Jelinek 1976](#), [Baker 1975a](#), [Bahl et al. 1983](#), [Jelinek 1990](#)). A trigram model was used in the IBM TANGORA speech recognition system in the 1970s, but the idea was not written up until later.

Add-one smoothing derives from Laplace’s 1812 law of succession and was first applied as an engineering solution to the zero-frequency problem by [Jeffreys \(1948\)](#)

based on an earlier Add-K suggestion by [Johnson \(1932\)](#). Problems with the add-one algorithm are summarized in [Gale and Church \(1994\)](#).

A wide variety of different language modeling and smoothing techniques were proposed in the 80s and 90s, including Good-Turing discounting—first applied to the n-gram smoothing at IBM by Katz ([Nádas 1984](#), [Church and Gale 1991](#))—Witten-Bell discounting ([Witten and Bell, 1991](#)), and varieties of **class-based n-gram** models that used information about word classes.

class-based n-gram

Starting in the late 1990s, Chen and Goodman produced a highly influential series of papers with a comparison of different language models ([Chen and Goodman 1996](#), [Chen and Goodman 1998](#), [Chen and Goodman 1999](#), [Goodman 2006](#)). They performed a number of carefully controlled experiments comparing different discounting algorithms, cache models, class-based models, and other language model parameters. They showed the advantages of **Modified Interpolated Kneser-Ney**, which has since become the standard baseline for language modeling, especially because they showed that caches and class-based models provided only minor additional improvement. These papers are recommended for any reader with further interest in language modeling.

Two commonly used toolkits for building language models are SRILM ([Stolcke, 2002](#)) and KenLM ([Heafield 2011](#), [Heafield et al. 2013](#)). Both are publicly available. SRILM offers a wider range of options and types of discounting, while KenLM is optimized for speed and memory size, making it possible to build web-scale language models.

The highest accuracy language models are **neural network** language models. These solve a major problem with n-gram language models: the number of parameters increases exponentially as the n-gram order increases, and n-grams have no way to generalize from training to test set. Neural language models instead project words into a **continuous** space in which words with similar contexts have similar representations. We'll introduce both **feedforward** language models ([Bengio et al. 2006](#), [Schwenk 2007](#)) in Chapter 7, and **recurrent** language models ([Mikolov, 2012](#)) in Chapter 9.

Exercises

- 3.1 Write out the equation for trigram probability estimation (modifying Eq. 3.11). Now write out all the non-zero trigram probabilities for the I am Sam corpus on page 33.
- 3.2 Calculate the probability of the sentence i want chinese food. Give two probabilities, one using Fig. 3.2 and the ‘useful probabilities’ just below it on page 35, and another using the add-1 smoothed table in Fig. 3.6. Assume the additional add-1 smoothed probabilities $P(i|<s>) = 0.19$ and $P(</s>|food) = 0.40$.
- 3.3 Which of the two probabilities you computed in the previous exercise is higher, unsmoothed or smoothed? Explain why.
- 3.4 We are given the following corpus, modified from the one in the chapter:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>
```

Using a bigram language model with add-one smoothing, what is $P(\text{Sam} \mid \text{am})$? Include $\langle s \rangle$ and $\langle /s \rangle$ in your counts just like any other token.

- 3.5** Suppose we didn't use the end-symbol $\langle /s \rangle$. Train an unsmoothed bigram grammar on the following training corpus without using the end-symbol $\langle /s \rangle$:

```
<s> a b
<s> b b
<s> b a
<s> a a
```

Demonstrate that your bigram model does not assign a single probability distribution across all sentence lengths by showing that the sum of the probability of the four possible 2 word sentences over the alphabet {a,b} is 1.0, and the sum of the probability of all possible 3 word sentences over the alphabet {a,b} is also 1.0.

- 3.6** Suppose we train a trigram language model with add-one smoothing on a given corpus. The corpus contains V word types. Express a formula for estimating $P(w_3|w_1, w_2)$, where w_3 is a word which follows the bigram (w_1, w_2) , in terms of various N -gram counts and V . Use the notation $c(w_1, w_2, w_3)$ to denote the number of times that trigram (w_1, w_2, w_3) occurs in the corpus, and so on for bigrams and unigrams.

- 3.7** We are given the following corpus, modified from the one in the chapter:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>
```

If we use linear interpolation smoothing between a maximum-likelihood bigram model and a maximum-likelihood unigram model with $\lambda_1 = \frac{1}{2}$ and $\lambda_2 = \frac{1}{2}$, what is $P(\text{Sam}|\text{am})$? Include $\langle s \rangle$ and $\langle /s \rangle$ in your counts just like any other token.

- 3.8** Write a program to compute unsmoothed unigrams and bigrams.
- 3.9** Run your n-gram program on two different small corpora of your choice (you might use email text or newsgroups). Now compare the statistics of the two corpora. What are the differences in the most common unigrams between the two? How about interesting differences in bigrams?
- 3.10** Add an option to your program to generate random sentences.
- 3.11** Add an option to your program to compute the perplexity of a test set.
- 3.12** Given a training set of 100 numbers consists of 91 zeros and 1 each of the other digits 1-9. Now we see the following test set: 0 0 0 0 0 3 0 0 0 0. What is the unigram perplexity?

CHAPTER

4

Naive Bayes and Sentiment Classification

Classification lies at the heart of both human and machine intelligence. Deciding what letter, word, or image has been presented to our senses, recognizing faces or voices, sorting mail, assigning grades to homeworks; these are all examples of assigning a category to an input. The potential challenges of this task are highlighted by the fabulist Jorge Luis Borges (1964), who imagined classifying animals into:

(a) those that belong to the Emperor, (b) embalmed ones, (c) those that are trained, (d) suckling pigs, (e) mermaids, (f) fabulous ones, (g) stray dogs, (h) those that are included in this classification, (i) those that tremble as if they were mad, (j) innumerable ones, (k) those drawn with a very fine camel's hair brush, (l) others, (m) those that have just broken a flower vase, (n) those that resemble flies from a distance.

Many language processing tasks involve classification, although luckily our classes are much easier to define than those of Borges. In this chapter we introduce the naive Bayes algorithm and apply it to **text categorization**, the task of assigning a label or category to an entire text or document.

text
categorization

sentiment
analysis

We focus on one common text categorization task, **sentiment analysis**, the extraction of **sentiment**, the positive or negative orientation that a writer expresses toward some object. A review of a movie, book, or product on the web expresses the author's sentiment toward the product, while an editorial or political text expresses sentiment toward a candidate or political action. Extracting consumer or public sentiment is thus relevant for fields from marketing to politics.

The simplest version of sentiment analysis is a binary classification task, and the words of the review provide excellent cues. Consider, for example, the following phrases extracted from positive and negative reviews of movies and restaurants. Words like *great, richly, awesome*, and *pathetic, awful* and *ridiculously* are very informative cues:

- + ...zany characters and richly applied satire, and some great plot twists
- It was pathetic. The worst part about it was the boxing scenes...
- + ...awesome caramel sauce and sweet toasty almonds. I love this place!
- ...awful pizza and ridiculously overpriced...

spam detection

Spam detection is another important commercial application, the binary classification task of assigning an email to one of the two classes *spam* or *not-spam*. Many lexical and other features can be used to perform this classification. For example you might quite reasonably be suspicious of an email containing phrases like “online pharmaceutical” or “WITHOUT ANY COST” or “Dear Winner”.

language id

authorship
attribution

Another thing we might want to know about a text is the language it's written in. Texts on social media, for example, can be in any number of languages and we'll need to apply different processing. The task of **language id** is thus the first step in most language processing pipelines. Related tasks like determining a text's author, (**authorship attribution**), or author characteristics like gender, age, and native

language are text classification tasks that are also relevant to the digital humanities, social sciences, and forensic linguistics.

Finally, one of the oldest tasks in text classification is assigning a library subject category or topic label to a text. Deciding whether a research paper concerns epidemiology or instead, perhaps, embryology, is an important component of information retrieval. Various sets of subject categories exist, such as the MeSH (Medical Subject Headings) thesaurus. In fact, as we will see, subject category classification is the task for which the naive Bayes algorithm was invented in 1961.

Classification is essential for tasks below the level of the document as well. We've already seen period disambiguation (deciding if a period is the end of a sentence or part of a word), and word tokenization (deciding if a character should be a word boundary). Even language modeling can be viewed as classification: each word can be thought of as a class, and so predicting the next word is classifying the context-so-far into a class for each next word. A part-of-speech tagger (Chapter 8) classifies each occurrence of a word in a sentence as, e.g., a noun or a verb.

The goal of classification is to take a single observation, extract some useful features, and thereby **classify** the observation into one of a set of discrete classes. One method for classifying text is to use handwritten rules. There are many areas of language processing where handwritten rule-based classifiers constitute a state-of-the-art system, or at least part of it.

**supervised
machine
learning**

Rules can be fragile, however, as situations or data change over time, and for some tasks humans aren't necessarily good at coming up with the rules. Most cases of classification in language processing are instead done via **supervised machine learning**, and this will be the subject of the remainder of this chapter. In supervised learning, we have a data set of input observations, each associated with some correct output (a 'supervision signal'). The goal of the algorithm is to learn how to map from a new observation to a correct output.

Formally, the task of supervised classification is to take an input x and a fixed set of output classes $Y = y_1, y_2, \dots, y_M$ and return a predicted class $y \in Y$. For text classification, we'll sometimes talk about c (for "class") instead of y as our output variable, and d (for "document") instead of x as our input variable. In the supervised situation we have a training set of N documents that have each been hand-labeled with a class: $(d_1, c_1), \dots, (d_N, c_N)$. Our goal is to learn a classifier that is capable of mapping from a new document d to its correct class $c \in C$. A **probabilistic classifier** additionally will tell us the probability of the observation being in the class. This full distribution over the classes can be useful information for downstream decisions; avoiding making discrete decisions early on can be useful when combining systems.

Many kinds of machine learning algorithms are used to build classifiers. This chapter introduces naive Bayes; the following one introduces logistic regression. These exemplify two ways of doing classification. **Generative** classifiers like naive Bayes build a model of how a class could generate some input data. Given an observation, they return the class most likely to have generated the observation. **Discriminative** classifiers like logistic regression instead learn what features from the input are most useful to discriminate between the different possible classes. While discriminative systems are often more accurate and hence more commonly used, generative classifiers still have a role.

4.1 Naive Bayes Classifiers

naive Bayes classifier In this section we introduce the **multinomial naive Bayes classifier**, so called because it is a Bayesian classifier that makes a simplifying (naive) assumption about how the features interact.

bag-of-words The intuition of the classifier is shown in Fig. 4.1. We represent a text document as if it were a **bag-of-words**, that is, an unordered set of words with their position ignored, keeping only their frequency in the document. In the example in the figure, instead of representing the word order in all the phrases like “I love this movie” and “I would recommend it”, we simply note that the word *I* occurred 5 times in the entire excerpt, the word *it* 6 times, the words *love*, *recommend*, and *movie* once, and so on.

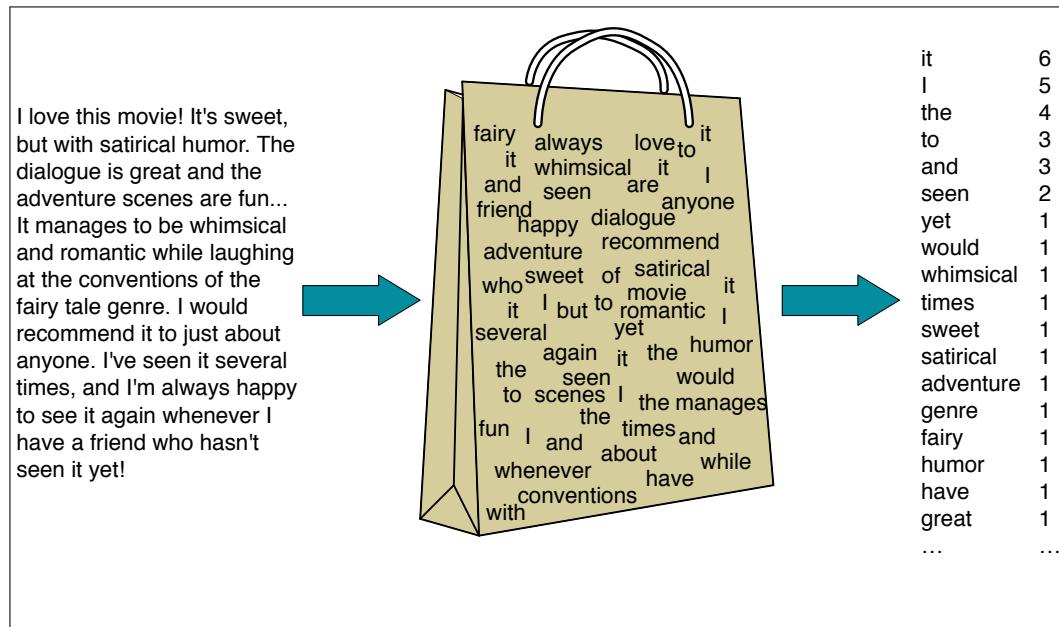


Figure 4.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Naive Bayes is a probabilistic classifier, meaning that for a document d , out of all classes $c \in C$ the classifier returns the class \hat{c} which has the maximum posterior probability given the document. In Eq. 4.1 we use the hat notation $\hat{\cdot}$ to mean “our estimate of the correct class”.

$$\hat{c} = \operatorname{argmax}_{c \in C} P(c|d) \quad (4.1)$$

Bayesian inference This idea of **Bayesian inference** has been known since the work of Bayes (1763), and was first applied to text classification by Mosteller and Wallace (1964). The intuition of Bayesian classification is to use Bayes’ rule to transform Eq. 4.1 into other probabilities that have some useful properties. Bayes’ rule is presented in Eq. 4.2; it gives us a way to break down any conditional probability $P(x|y)$ into three other probabilities:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \quad (4.2)$$

We can then substitute Eq. 4.2 into Eq. 4.1 to get Eq. 4.3:

$$\hat{c} = \operatorname{argmax}_{c \in C} P(c|d) = \operatorname{argmax}_{c \in C} \frac{P(d|c)P(c)}{P(d)} \quad (4.3)$$

We can conveniently simplify Eq. 4.3 by dropping the denominator $P(d)$. This is possible because we will be computing $\frac{P(d|c)P(c)}{P(d)}$ for each possible class. But $P(d)$ doesn't change for each class; we are always asking about the most likely class for the same document d , which must have the same probability $P(d)$. Thus, we can choose the class that maximizes this simpler formula:

$$\hat{c} = \operatorname{argmax}_{c \in C} P(c|d) = \operatorname{argmax}_{c \in C} P(d|c)P(c) \quad (4.4)$$

We thus compute the most probable class \hat{c} given some document d by choosing the class which has the highest product of two probabilities: the **prior probability** of the class $P(c)$ and the **likelihood** of the document $P(d|c)$:

$$\hat{c} = \operatorname{argmax}_{c \in C} \underbrace{P(d|c)}_{\text{likelihood}} \underbrace{P(c)}_{\text{prior}} \quad (4.5)$$

Without loss of generalization, we can represent a document d as a set of features f_1, f_2, \dots, f_n :

$$\hat{c} = \operatorname{argmax}_{c \in C} \underbrace{P(f_1, f_2, \dots, f_n|c)}_{\text{likelihood}} \underbrace{P(c)}_{\text{prior}} \quad (4.6)$$

Unfortunately, Eq. 4.6 is still too hard to compute directly: without some simplifying assumptions, estimating the probability of every possible combination of features (for example, every possible set of words and positions) would require huge numbers of parameters and impossibly large training sets. Naive Bayes classifiers therefore make two simplifying assumptions.

The first is the *bag of words* assumption discussed intuitively above: we assume position doesn't matter, and that the word "love" has the same effect on classification whether it occurs as the 1st, 20th, or last word in the document. Thus we assume that the features f_1, f_2, \dots, f_n only encode word identity and not position.

The second is commonly called the **naive Bayes assumption**: this is the conditional independence assumption that the probabilities $P(f_i|c)$ are independent given the class c and hence can be 'naively' multiplied as follows:

$$P(f_1, f_2, \dots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \dots \cdot P(f_n|c) \quad (4.7)$$

The final equation for the class chosen by a naive Bayes classifier is thus:

$$c_{NB} = \operatorname{argmax}_{c \in C} P(c) \prod_{f \in F} P(f|c) \quad (4.8)$$

To apply the naive Bayes classifier to text, we need to consider word positions, by simply walking an index through every word position in the document:

$$\begin{aligned} \text{positions} &\leftarrow \text{all word positions in test document} \\ c_{NB} &= \operatorname{argmax}_{c \in C} P(c) \prod_{i \in \text{positions}} P(w_i|c) \end{aligned} \quad (4.9)$$

Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed. Thus Eq. 4.9 is generally instead expressed as

$$c_{NB} = \operatorname{argmax}_{c \in C} \log P(c) + \sum_{i \in \text{positions}} \log P(w_i | c) \quad (4.10)$$

linear classifiers

By considering features in log space, Eq. 4.10 computes the predicted class as a linear function of input features. Classifiers that use a linear combination of the inputs to make a classification decision —like naive Bayes and also logistic regression— are called **linear classifiers**.

4.2 Training the Naive Bayes Classifier

How can we learn the probabilities $P(c)$ and $P(f_i|c)$? Let's first consider the maximum likelihood estimate. We'll simply use the frequencies in the data. For the document prior $P(c)$ we ask what percentage of the documents in our training set are in each class c . Let N_c be the number of documents in our training data with class c and N_{doc} be the total number of documents. Then:

$$\hat{P}(c) = \frac{N_c}{N_{doc}} \quad (4.11)$$

To learn the probability $P(f_i|c)$, we'll assume a feature is just the existence of a word in the document's bag of words, and so we'll want $P(w_i|c)$, which we compute as the fraction of times the word w_i appears among all words in all documents of topic c . We first concatenate all documents with category c into one big “category c ” text. Then we use the frequency of w_i in this concatenated document to give a maximum likelihood estimate of the probability:

$$\hat{P}(w_i | c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)} \quad (4.12)$$

Here the vocabulary V consists of the union of all the word types in all classes, not just the words in one class c .

There is a problem, however, with maximum likelihood training. Imagine we are trying to estimate the likelihood of the word “fantastic” given class *positive*, but suppose there are no training documents that both contain the word “fantastic” and are classified as *positive*. Perhaps the word “fantastic” happens to occur (sarcastically?) in the class *negative*. In such a case the probability for this feature will be zero:

$$\hat{P}(\text{“fantastic”} | \text{positive}) = \frac{\text{count(“fantastic”, positive)}}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0 \quad (4.13)$$

But since naive Bayes naively multiplies all the feature likelihoods together, zero probabilities in the likelihood term for any class will cause the probability of the class to be zero, no matter the other evidence!

The simplest solution is the add-one (Laplace) smoothing introduced in Chapter 3. While Laplace smoothing is usually replaced by more sophisticated smoothing

algorithms in language modeling, it is commonly used in naive Bayes text categorization:

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V} (\text{count}(w, c) + 1)} = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V} \text{count}(w, c)) + |V|} \quad (4.14)$$

Note once again that it is crucial that the vocabulary V consists of the union of all the word types in all classes, not just the words in one class c (try to convince yourself why this must be true; see the exercise at the end of the chapter).

unknown word

stop words

What do we do about words that occur in our test data but are not in our vocabulary at all because they did not occur in any training document in any class? The solution for such **unknown words** is to ignore them—remove them from the test document and not include any probability for them at all.

Finally, some systems choose to completely ignore another class of words: **stop words**, very frequent words like *the* and *a*. This can be done by sorting the vocabulary by frequency in the training set, and defining the top 10–100 vocabulary entries as stop words, or alternatively by using one of the many pre-defined stop word list available online. Then every instance of these stop words are simply removed from both training and test documents as if they had never occurred. In most text classification applications, however, using a stop word list doesn't improve performance, and so it is more common to make use of the entire vocabulary and not use a stop word list.

Fig. 4.2 shows the final algorithm.

```

function TRAIN NAIVE BAYES(D, C) returns log  $P(c)$  and log  $P(w|c)$ 
  for each class  $c \in C$            # Calculate  $P(c)$  terms
     $N_{doc}$  = number of documents in D
     $N_c$  = number of documents from D in class c
     $logprior[c] \leftarrow \log \frac{N_c}{N_{doc}}$ 
     $V \leftarrow$  vocabulary of D
     $bigdoc[c] \leftarrow \text{append}(d)$  for  $d \in D$  with class  $c$ 
      for each word  $w$  in  $V$            # Calculate  $P(w|c)$  terms
         $count(w, c) \leftarrow$  # of occurrences of  $w$  in  $bigdoc[c]$ 
         $loglikelihood[w, c] \leftarrow \log \frac{count(w, c) + 1}{\sum_{w' \text{ in } V} (count(w', c) + 1)}$ 
  return  $logprior, loglikelihood, V$ 

function TEST NAIVE BAYES( $testdoc, logprior, loglikelihood, C, V$ ) returns best  $c$ 
  for each class  $c \in C$ 
     $sum[c] \leftarrow logprior[c]$ 
    for each position  $i$  in  $testdoc$ 
       $word \leftarrow testdoc[i]$ 
      if  $word \in V$ 
         $sum[c] \leftarrow sum[c] + loglikelihood[word, c]$ 
  return  $\text{argmax}_c sum[c]$ 
```

Figure 4.2 The naive Bayes algorithm, using add-1 smoothing. To use add- α smoothing instead, change the $+1$ to $+\alpha$ for loglikelihood counts in training.

4.3 Worked example

Let's walk through an example of training and testing naive Bayes with add-one smoothing. We'll use a sentiment analysis domain with the two classes positive (+) and negative (-), and take the following miniature training and test documents simplified from actual movie reviews.

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

The prior $P(c)$ for the two classes is computed via Eq. 4.11 as $\frac{N_c}{N_{doc}}$:

$$P(-) = \frac{3}{5} \quad P(+) = \frac{2}{5}$$

The word *with* doesn't occur in the training set, so we drop it completely (as mentioned above, we don't use unknown word models for naive Bayes). The likelihoods from the training set for the remaining three words "predictable", "no", and "fun", are as follows, from Eq. 4.14 (computing the probabilities for the remainder of the words in the training set is left as an exercise for the reader):

$$\begin{aligned} P(\text{"predictable"}|-) &= \frac{1+1}{14+20} & P(\text{"predictable"}|+) &= \frac{0+1}{9+20} \\ P(\text{"no"}|-) &= \frac{1+1}{14+20} & P(\text{"no"}|+) &= \frac{0+1}{9+20} \\ P(\text{"fun"}|-) &= \frac{0+1}{14+20} & P(\text{"fun"}|+) &= \frac{1+1}{9+20} \end{aligned}$$

For the test sentence S = "predictable with no fun", after removing the word 'with', the chosen class, via Eq. 4.9, is therefore computed as follows:

$$\begin{aligned} P(-)P(S|-) &= \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5} \\ P(+)P(S|+) &= \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5} \end{aligned}$$

The model thus predicts the class *negative* for the test sentence.

4.4 Optimizing for Sentiment Analysis

While standard naive Bayes text classification can work well for sentiment analysis, some small changes are generally employed that improve performance.

First, for sentiment classification and a number of other text classification tasks, whether a word occurs or not seems to matter more than its frequency. Thus it often improves performance to clip the word counts in each document at 1 (see the end of the chapter for pointers to these results). This variant is called **binary**

binary NB **multinomial naive Bayes** or **binary NB**. The variant uses the same Eq. 4.10 except that for each document we remove all duplicate words before concatenating them into the single big document. Fig. 4.3 shows an example in which a set of four documents (shortened and text-normalized for this example) are remapped to binary, with the modified counts shown in the table on the right. The example is worked without add-1 smoothing to make the differences clearer. Note that the results counts need not be 1; the word *great* has a count of 2 even for Binary NB, because it appears in multiple documents.

	NB Counts		Binary Counts	
	+	-	+	-
Four original documents:				
– it was pathetic the worst part was the boxing scenes	and	2	0	1 0
– no plot twists or great scenes	boxing	0	1	0 1
+ and satire and great plot twists	film	1	0	1 0
+ great scenes great film	great	3	1	2 1
After per-document binarization:				
– it was pathetic the worst part boxing scenes	it	0	1	0 1
– no plot twists or great scenes	no	0	1	0 1
+ and satire great plot twists	or	0	1	0 1
+ great scenes film	part	0	1	0 1
	pathetic	0	1	0 1
	plot	1	1	1 1
	satire	1	0	1 0
	scenes	1	2	1 2
	the	0	2	0 1
	twists	1	1	1 1
	was	0	2	0 1
	worst	0	1	0 1

Figure 4.3 An example of binarization for the binary naive Bayes algorithm.

A second important addition commonly made when doing text classification for sentiment is to deal with negation. Consider the difference between *I really like this movie* (positive) and *I didn't like this movie* (negative). The negation expressed by *didn't* completely alters the inferences we draw from the predicate *like*. Similarly, negation can modify a negative word to produce a positive review (*don't dismiss this film, doesn't let us get bored*).

A very simple baseline that is commonly used in sentiment analysis to deal with negation is the following: during text normalization, prepend the prefix *NOT_* to every word after a token of logical negation (*n't, not, no, never*) until the next punctuation mark. Thus the phrase

didn't like this movie , but I
becomes

didn't NOT_like NOT_this NOT_movie , but I

Newly formed ‘words’ like *NOT_like*, *NOT_recommend* will thus occur more often in negative document and act as cues for negative sentiment, while words like *NOT_bored*, *NOT_dismiss* will acquire positive associations. We will return in Chapter 17 to the use of parsing to deal more accurately with the scope relationship between these negation words and the predicates they modify, but this simple baseline works quite well in practice.

Finally, in some situations we might have insufficient labeled training data to train accurate naive Bayes classifiers using all words in the training set to estimate positive and negative sentiment. In such cases we can instead derive the positive

sentiment lexicons
General Inquirer
LIWC

and negative word features from **sentiment lexicons**, lists of words that are pre-annotated with positive or negative sentiment. Four popular lexicons are the **General Inquirer** (Stone et al., 1966), **LIWC** (Pennebaker et al., 2007), the opinion lexicon of **Hu and Liu** (2004a) and the MPQA Subjectivity Lexicon (Wilson et al., 2005).

For example the MPQA subjectivity lexicon has 6885 words, 2718 positive and 4912 negative, each marked for whether it is strongly or weakly biased. (Chapter 21 will discuss how these lexicons can be learned automatically.) Some samples of positive and negative words from the MPQA lexicon include:

- + : *admirable, beautiful, confident, dazzling, ecstatic, favor, glee, great*
- : *awful, bad, bias, catastrophe, cheat, deny, envious, foul, harsh, hate*

A common way to use lexicons in a naive Bayes classifier is to add a feature that is counted whenever a word from that lexicon occurs. Thus we might add a feature called ‘this word occurs in the positive lexicon’, and treat all instances of words in the lexicon as counts for that one feature, instead of counting each word separately. Similarly, we might add as a second feature ‘this word occurs in the negative lexicon’ of words in the negative lexicon. If we have lots of training data, and if the test data matches the training data, using just two features won’t work as well as using all the words. But when training data is sparse or not representative of the test set, using dense lexicon features instead of sparse individual-word features may generalize better.

4.5 Naive Bayes for other text classification tasks

spam detection

In the previous section we pointed out that naive Bayes doesn’t require that our classifier use all the words in the training data as features. In fact features in naive Bayes can express any property of the input text we want.

Consider the task of **spam detection**, deciding if a particular piece of email is an example of spam (unsolicited bulk email) — and one of the first applications of naive Bayes to text classification (Sahami et al., 1998).

A common solution here, rather than using all the words as individual features, is to predefine likely sets of words or phrases as features, combined these with features that are not purely linguistic. For example the open-source SpamAssassin tool¹ defines features like the phrase “one hundred percent guaranteed”, or the feature *mentions millions of dollars*, which is a regular expression that matches suspiciously large sums of money. But it also includes features like *HTML has a low ratio of text to image area*, that isn’t purely linguistic and might require some sophisticated computation, or totally non-linguistic features about, say, the path that the email took to arrive. More sample SpamAssassin features:

- Email subject line is all capital letters
- Contains phrases of urgency like “urgent reply”
- Email subject line contains “online pharmaceutical”
- HTML has unbalanced ”head” tags
- Claims you can be removed from the list

language ID

For other tasks, like **language ID**—determining what language a given piece of text is written in—the most effective naive Bayes features are not words at all, but **byte n-grams**, 2-grams (‘zw’) 3-grams (‘nya’, ‘Vo’), or 4-grams (‘ie z’, ‘thei’).

¹ <https://spamassassin.apache.org>

Because spaces count as a byte, byte n-grams can model statistics about the beginning or ending of words.² A widely used naive Bayes system, `langid.py` ([Lui and Baldwin, 2012](#)) begins with all possible n-grams of lengths 1-4, using **feature selection** to winnow down to the most informative 7000 final features.

Language ID systems are trained on multilingual text, such as Wikipedia (Wikipedia text in 68 different languages were used in ([Lui and Baldwin, 2011](#))), or newswire. To make sure that this multilingual text correctly reflects different regions, dialects, and socioeconomic classes, systems also add Twitter text in many languages geo-tagged to many regions (important for getting world English dialects from countries with large Anglophone populations like Nigeria or India), Bible and Quran translations, slang websites like Urban Dictionary, corpora of African American Vernacular English ([Blodgett et al., 2016](#)), and so on ([Jurgens et al., 2017](#)).

4.6 Naive Bayes as a Language Model

As we saw in the previous section, naive Bayes classifiers can use any sort of feature: dictionaries, URLs, email addresses, network features, phrases, and so on. But if, as in the previous section, we use only individual word features, and we use all of the words in the text (not a subset), then naive Bayes has an important similarity to language modeling. Specifically, a naive Bayes model can be viewed as a set of class-specific unigram language models, in which the model for each class instantiates a unigram language model.

Since the likelihood features from the naive Bayes model assign a probability to each word $P(\text{word}|c)$, the model also assigns a probability to each sentence:

$$P(s|c) = \prod_{i \in \text{positions}} P(w_i|c) \quad (4.15)$$

Thus consider a naive Bayes model with the classes *positive* (+) and *negative* (-) and the following model parameters:

w	$P(w +)$	$P(w -)$
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...

Each of the two columns above instantiates a language model that can assign a probability to the sentence “I love this fun film”:

$$P(\text{"I love this fun film"}|+) = 0.1 \times 0.1 \times 0.01 \times 0.05 \times 0.1 = 0.0000005$$

$$P(\text{"I love this fun film"}|-) = 0.2 \times 0.001 \times 0.01 \times 0.005 \times 0.1 = .000000010$$

As it happens, the positive model assigns a higher probability to the sentence: $P(s|pos) > P(s|neg)$. Note that this is just the likelihood part of the naive Bayes

² It's also possible to use codepoints, which are multi-byte Unicode representations of characters in character sets, but simply using bytes seems to work better.

model; once we multiply in the prior a full naive Bayes model might well make a different classification decision.

4.7 Evaluation: Precision, Recall, F-measure

To introduce the methods for evaluating text classification, let's first consider some simple binary *detection* tasks. For example, in spam detection, our goal is to label every text as being in the spam category ("positive") or not in the spam category ("negative"). For each item (email document) we therefore need to know whether our system called it spam or not. We also need to know whether the email is actually spam or not, i.e. the human-defined labels for each document that we are trying to match. We will refer to these human labels as the **gold labels**.

Or imagine you're the CEO of the *Delicious Pie Company* and you need to know what people are saying about your pies on social media, so you build a system that detects tweets concerning Delicious Pie. Here the positive class is tweets about Delicious Pie and the negative class is all other tweets.

In both cases, we need a metric for knowing how well our spam detector (or pie-tweet-detector) is doing. To evaluate any system for detecting things, we start by building a **contingency table** like the one shown in Fig. 4.4. Each cell labels a set of possible outcomes. In the spam detection case, for example, true positives are documents that are indeed spam (indicated by human-created gold labels) and our system said they were spam. False negatives are documents that are indeed spam but our system labeled as non-spam.

To the bottom right of the table is the equation for *accuracy*, which asks what percentage of all the observations (for the spam or pie examples that means all emails or tweets) our system labeled correctly. Although accuracy might seem a natural metric, we generally don't use it. That's because accuracy doesn't work well when the classes are unbalanced (as indeed they are with spam, which is a large majority of email, or with tweets, which are mainly not about pie).

		<i>gold standard labels</i>			
		gold positive	gold negative		
		true positive	false positive	precision =	$\frac{tp}{tp+fp}$
system output labels	system positive				
	system negative	false negative	true negative	recall =	$\frac{tp}{tp+fn}$
				accuracy =	$\frac{tp+tn}{tp+fp+tn+fn}$

Figure 4.4 Contingency table

To make this more explicit, imagine that we looked at a million tweets, and let's say that only 100 of them are discussing their love (or hatred) for our pie, while the other 999,900 are tweets about something completely unrelated. Imagine a simple classifier that stupidly classified every tweet as "not about pie". This classifier would have 999,900 true negatives and only 100 false negatives for an accuracy of $999,900/1,000,000$ or 99.99%! What an amazing accuracy level! Surely we should be happy with this classifier? But of course this fabulous 'no pie' classifier would

be completely useless, since it wouldn't find a single one of the customer comments we are looking for. In other words, accuracy is not a good metric when the goal is to discover something that is rare, or at least not completely balanced in frequency, which is a very common situation in the world.

precision That's why instead of accuracy we generally turn to two other metrics: **precision** and **recall**. **Precision** measures the percentage of the items that the system detected (i.e., the system labeled as positive) that are in fact positive (i.e., are positive according to the human gold labels). Precision is defined as

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

recall **Recall** measures the percentage of items actually present in the input that were correctly identified by the system. Recall is defined as

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Precision and recall will help solve the problem with the useless “nothing is pie” classifier. This classifier, despite having a fabulous accuracy of 99.99%, has a terrible recall of 0 (since there are no true positives, and 100 false negatives, the recall is 0/100). You should convince yourself that the precision at finding relevant tweets is equally problematic. Thus precision and recall, unlike accuracy, emphasize true positives: finding the things that we are supposed to be looking for.

F-measure There are many ways to define a single metric that incorporates aspects of both precision and recall. The simplest of these combinations is the **F-measure** ([van Rijsbergen, 1975](#)), defined as:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

The β parameter differentially weights the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is the most frequently used metric, and is called $F_{\beta=1}$ or just F_1 :

$$F_1 = \frac{2PR}{P+R} \quad (4.16)$$

F-measure comes from a weighted harmonic mean of precision and recall. The harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of reciprocals:

$$\text{HarmonicMean}(a_1, a_2, a_3, a_4, \dots, a_n) = \frac{n}{\frac{1}{a_1} + \frac{1}{a_2} + \frac{1}{a_3} + \dots + \frac{1}{a_n}} \quad (4.17)$$

and hence F-measure is

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} \quad \text{or} \left(\text{with } \beta^2 = \frac{1 - \alpha}{\alpha} \right) \quad F = \frac{(\beta^2 + 1)PR}{\beta^2P + R} \quad (4.18)$$

Harmonic mean is used because it is a conservative metric; the harmonic mean of two values is closer to the minimum of the two values than the arithmetic mean is. Thus it weighs the lower of the two numbers more heavily.

4.7.1 More than two classes

Up to now we have been assuming text classification tasks with only two classes. But lots of classification tasks in language processing have more than two classes. For sentiment analysis we generally have 3 classes (positive, negative, neutral) and even more classes are common for tasks like part-of-speech tagging, word sense disambiguation, semantic role labeling, emotion detection, and so on.

any-of There are two kinds of multi-class classification tasks. In **any-of** or **multi-label classification**, each document or item can be assigned more than one label. We can solve *any-of* classification by building separate binary classifiers for each class c , trained on positive examples labeled c and negative examples not labeled c . Given a test document or item d , then each classifier makes their decision independently, and we may assign multiple labels to d .

one-of multinomial classification More common in language processing is **one-of** or **multinomial classification**, in which the classes are mutually exclusive and each document or item appears in exactly one class. Here we again build a separate binary classifier trained on positive examples from c and negative examples from all other classes. Now given a test document or item d , we run all the classifiers and choose the label from the classifier with the highest score. Consider the sample confusion matrix for a hypothetical 3-way *one-of* email categorization decision (urgent, normal, spam) shown in Fig. 4.5.

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recalls} = \frac{200}{1+50+200}$	

Figure 4.5 Confusion matrix for a three-class categorization task, showing for each pair of classes (c_1, c_2) , how many documents from c_1 were (in)correctly assigned to c_2

macroaveraging The matrix shows, for example, that the system mistakenly labeled 1 spam document as urgent, and we have shown how to compute a distinct precision and recall value for each class. In order to derive a single metric that tells us how well the system is doing, we can combine these values in two ways. In **macroaveraging**, we compute the performance for each class, and then average over classes. In **microaveraging**, we collect the decisions for all classes into a single contingency table, and then compute precision and recall from that table. Fig. 4.6 shows the contingency table for each class separately, and shows the computation of microaveraged and macroaveraged precision.

microaveraging As the figure shows, a microaverage is dominated by the more frequent class (in this case spam), since the counts are pooled. The macroaverage better reflects the statistics of the smaller classes, and so is more appropriate when performance on all the classes is equally important.

Class 1: Urgent		Class 2: Normal		Class 3: Spam		Pooled									
		true	true	true	true	true	true								
system	urgent	8	11	system	normal	60	55	system	spam	200	33	system	yes	268	99
system	not	8	340	system	not	40	212	system	not	51	83	system	no	99	635
$\text{precision} = \frac{8}{8+11} = .42$		$\text{precision} = \frac{60}{60+55} = .52$		$\text{precision} = \frac{200}{200+33} = .86$		$\text{microaverage precision} = \frac{268}{268+99} = .73$									
$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = .60$															

Figure 4.6 Separate contingency tables for the 3 classes from the previous figure, showing the pooled contingency table and the microaveraged and macroaveraged precision.

4.8 Test sets and Cross-validation

development test set
devset

cross-validation

10-fold cross-validation

The training and testing procedure for text classification follows what we saw with language modeling (Section 3.2): we use the training set to train the model, then use the **development test set** (also called a **devset**) to perhaps tune some parameters, and in general decide what the best model is. Once we come up with what we think is the best model, we run it on the (hitherto unseen) test set to report its performance.

While the use of a devset avoids overfitting the test set, having a fixed training set, devset, and test set creates another problem: in order to save lots of data for training, the test set (or devset) might not be large enough to be representative. It would be better if we could somehow use **all** our data both for training and test. We do this by **cross-validation**: we randomly choose a training and test set division of our data, train our classifier, and then compute the error rate on the test set. Then we repeat with a different randomly selected training set and test set. We do this sampling process 10 times and average these 10 runs to get an average error rate. This is called **10-fold cross-validation**.

The only problem with cross-validation is that because all the data is used for testing, we need the whole corpus to be blind; we can't examine any of the data to suggest possible features and in general see what's going on. But looking at the corpus is often important for designing the system. For this reason, it is common to create a fixed training set and test set, then do 10-fold cross-validation inside the training set, but compute error rate the normal way in the test set, as shown in Fig. 4.7.

4.9 Statistical Significance Testing

In building systems we are constantly comparing the performance of systems. Often we have added some new bells and whistles to our algorithm and want to compare the new version of the system to the unaugmented version. Or we want to compare our algorithm to a previously published one to know which is better.

We might imagine that to compare the performance of two classifiers A and B all we have to do is look at A and B's score on the same test set—for example we might choose to compare macro-averaged F1—and see whether it's A or B that has



Figure 4.7 10-fold cross-validation

the higher score. But just looking at this one difference isn't good enough, because A might have a better performance than B on a particular test set just by chance.

Let's say we have a test set x of n observations $x = x_1, x_2, \dots, x_n$ on which A's performance is better than B by $\delta(x)$. How can we know if A is really better than B? To do so we'd need to reject the **null hypothesis** that A isn't really better than B and this difference $\delta(x)$ occurred purely by chance. If the null hypothesis was correct, we would expect that if we had many test sets of size n and we measured A and B's performance on all of them, that on average A might accidentally still be better than B by this amount $\delta(x)$ just by chance.

More formally, if we had a random variable X ranging over test sets, the null hypothesis H_0 expects $P(\delta(X) > \delta(x)|H_0)$, the probability that we'll see similarly big differences just by chance, to be high.

If we had all these test sets we could just measure all the $\delta(x')$ for all the x' . If we found that those deltas didn't seem to be bigger than $\delta(x)$, that is, that $p\text{-value}(x)$ was sufficiently small, less than the standard thresholds of 0.05 or 0.01, then we might reject the null hypothesis and agree that $\delta(x)$ was a sufficiently surprising difference and A is really a better algorithm than B. Following Berg-Kirkpatrick et al. (2012) we'll refer to $P(\delta(X) > \delta(x)|H_0)$ as $p\text{-value}(x)$.

In language processing we don't generally use traditional statistical approaches like paired t-tests to compare system outputs because most metrics are not normally distributed, violating the assumptions of the tests. The standard approach to computing $p\text{-value}(x)$ in natural language processing is to use non-parametric tests like the **bootstrap test** (Efron and Tibshirani, 1993)—which we will describe below—or a similar test, **approximate randomization** (Noreen, 1989). The advantage of these tests is that they can apply to any metric; from precision, recall, or F1 to the BLEU metric used in machine translation.

bootstrap test
approximate randomization

bootstrapping

The word **bootstrapping** refers to repeatedly drawing large numbers of smaller samples with replacement (called **bootstrap samples**) from an original larger sample. The intuition of the bootstrap test is that we can create many virtual test sets from an observed test set by repeatedly sampling from it. The method only makes the assumption that the sample is representative of the population.

Consider a tiny text classification example with a test set x of 10 documents. The first row of Fig. 4.8 shows the results of two classifiers (A and B) on this test set, with each document labeled by one of the four possibilities: (A and B both right, both wrong, A right and B wrong, A wrong and B right); a slash through a letter

(B) means that that classifier got the answer wrong. On the first document both A and B get the correct class (AB), while on the second document A got it right but B got it wrong (A~~B~~). If we assume for simplicity that our metric is accuracy, A has an accuracy of .70 and B of .50, so $\delta(x)$ is .20. To create each virtual test set of size $N = 10$, we repeatedly (10 times) select a cell from row x with replacement. Fig. 4.8 shows a few examples.

	1	2	3	4	5	6	7	8	9	10	A%	B%	$\delta()$
x	AB	AB B	AB	AB A	AB	AB A	AB	AB A	AB	AB A	.70	.50	.20
$x^{*(1)}$	AB A	AB	AB A	AB A	AB	AB A	AB	AB A	AB	AB	.60	.60	.00
$x^{*(2)}$	AB B	AB	AB A	AB A	AB	AB A	AB	AB A	AB	AB	.60	.70	-.10
...													
$x^{*(b)}$													

Figure 4.8 The bootstrap: Examples of b pseudo test sets being created from an initial true test set x . Each pseudo test set is created by sampling $n = 10$ times with replacement; thus an individual sample is a single cell, a document with its gold label and the correct or incorrect performance of classifiers A and B.

Now that we have a sampling distribution, we can do statistics on how often A has an accidental advantage. There are various ways to compute this advantage; here we follow the version laid out in Berg-Kirkpatrick et al. (2012). We might think that we should just ask, for each bootstrap sample $x^{*(i)}$, whether A beats B by more than $\delta(x)$. But there's a problem: we didn't draw these samples from a distribution with 0 mean. The $x^{*(i)}$ were sampled from x , and so the expected value of $\delta(x^{*(i)})$ lies very close to $\delta(x)$. That is, about half the time A will be better than B, so we *expect* A to beat B by $\delta(x)$. Instead, we want to know how often A beats these expectations by more than $\delta(x)$. To correct for the expected success, we need to zero-center, subtracting $\delta(x)$ from each pseudo test set. Thus we'll be comparing for each $x^{*(i)}$ whether $\delta(x^{*(i)}) > 2\delta(x)$. The full algorithm for the bootstrap is shown in Fig. 4.9. It is given a test set x , a number of samples b , and counts the percentage of the b bootstrap test sets in which $\delta(x^{*(i)}) > 2\delta(x)$. This percentage then acts as a one-sided empirical p-value (more sophisticated ways to get p-values from confidence intervals also exist).

```

function BOOTSTRAP(test set  $x$ , num of samples  $b$ ) returns  $p\text{-value}(x)$ 
    Calculate  $\delta(x)$  # how much better does algorithm A do than B on  $x$ 
    for  $i = 1$  to  $b$  do
        for  $j = 1$  to  $n$  do # Draw a bootstrap sample  $x^{*(i)}$  of size n
            Select a member of  $x$  at random and add it to  $x^{*(i)}$ 
        Calculate  $\delta(x^{*(i)})$  # how much better does algorithm A do than B on  $x^{*(i)}$ 
        for each  $x^{*(i)}$ 
             $s \leftarrow s + 1$  if  $\delta(x^{*(i)}) > 2\delta(x)$ 
     $p\text{-value}(x) \approx \frac{s}{b}$  # on what % of the b samples did algorithm A beat expectations?
    return  $p\text{-value}(x)$ 

```

Figure 4.9 A version of the bootstrap algorithm after Berg-Kirkpatrick et al. (2012).

4.10 Summary

This chapter introduced the **naive Bayes** model for **classification** and applied it to the **text categorization** task of **sentiment analysis**.

- Many language processing tasks can be viewed as tasks of **classification**.
- Text categorization, in which an entire text is assigned a class from a finite set, includes such tasks as **sentiment analysis**, **spam detection**, language identification, and authorship attribution.
- Sentiment analysis classifies a text as reflecting the positive or negative orientation (**sentiment**) that a writer expresses toward some object.
- Naive Bayes is a **generative** model that makes the bag of words assumption (position doesn't matter) and the conditional independence assumption (words are conditionally independent of each other given the class)
- Naive Bayes with binarized features seems to work better for many text classification tasks.
- Classifiers are evaluated based on **precision** and **recall**.
- Classifiers are trained using distinct training, dev, and test sets, including the use of **cross-validation** in the training set.

Bibliographical and Historical Notes

Multinomial naive Bayes text classification was proposed by [Maron \(1961\)](#) at the RAND Corporation for the task of assigning subject categories to journal abstracts. His model introduced most of the features of the modern form presented here, approximating the classification task with one-of categorization, and implementing add- δ smoothing and information-based feature selection.

The conditional independence assumptions of naive Bayes and the idea of Bayesian analysis of text seems to have arisen multiple times. The same year as Maron's paper, [Minsky \(1961\)](#) proposed a naive Bayes classifier for vision and other artificial intelligence problems, and Bayesian techniques were also applied to the text classification task of authorship attribution by [Mosteller and Wallace \(1963\)](#). It had long been known that Alexander Hamilton, John Jay, and James Madison wrote the anonymously-published *Federalist* papers in 1787–1788 to persuade New York to ratify the United States Constitution. Yet although some of the 85 essays were clearly attributable to one author or another, the authorship of 12 were in dispute between Hamilton and Madison. [Mosteller and Wallace \(1963\)](#) trained a Bayesian probabilistic model of the writing of Hamilton and another model on the writings of Madison, then computed the maximum-likelihood author for each of the disputed essays. Naive Bayes was first applied to spam detection in [Heckerman et al. \(1998\)](#).

[Metsis et al. \(2006\)](#), [Pang et al. \(2002\)](#), and [Wang and Manning \(2012\)](#) show that using boolean attributes with multinomial naive Bayes works better than full counts. Binary multinomial naive Bayes is sometimes confused with another variant of naive Bayes that also use a binary representation of whether a term occurs in a document: **Multivariate Bernoulli naive Bayes**. The Bernoulli variant instead estimates $P(w|c)$ as the fraction of documents that contain a term, and includes a probability for whether a term is *not* in a document. [McCallum and Nigam \(1998\)](#) and [Wang and Manning \(2012\)](#) show that the multivariate Bernoulli variant of naive

Bayes doesn't work as well as the multinomial algorithm for sentiment or other text tasks.

There are a variety of sources covering the many kinds of text classification tasks. For sentiment analysis see [Pang and Lee \(2008\)](#), and [Liu and Zhang \(2012\)](#). [Stamatatos \(2009\)](#) surveys authorship attribute algorithms. On language identification see [Jauhainen et al. \(2018\)](#); [Jaech et al. \(2016\)](#) is an important early neural system. The task of newswire indexing was often used as a test case for text classification algorithms, based on the Reuters-21578 collection of newswire articles.

See [Manning et al. \(2008\)](#) and [Aggarwal and Zhai \(2012\)](#) on text classification; classification in general is covered in machine learning textbooks ([Hastie et al. 2001](#), [Witten and Frank 2005](#), [Bishop 2006](#), [Murphy 2012](#)).

Non-parametric methods for computing statistical significance were used first in NLP in the MUC competition ([Chinchor et al., 1993](#)), and even earlier in speech recognition ([Gillick and Cox 1989](#), [Bisani and Ney 2004](#)). Our description of the bootstrap draws on the description in [Berg-Kirkpatrick et al. \(2012\)](#). Recent work has focused on issues including multiple test sets and multiple metrics ([Søgaard et al. 2014](#), [Dror et al. 2017](#)).

information
gain

Feature selection is a method of removing features that are unlikely to generalize well. Features are generally ranked by how informative they are about the classification decision. A very common metric, **information gain**, tells us how many bits of information the presence of the word gives us for guessing the class. Other feature selection metrics include χ^2 , pointwise mutual information, and GINI index; see [Yang and Pedersen \(1997\)](#) for a comparison and [Guyon and Elisseeff \(2003\)](#) for an introduction to feature selection.

Exercises

- 4.1** Assume the following likelihoods for each word being part of a positive or negative movie review, and equal prior probabilities for each class.

	pos	neg
I	0.09	0.16
always	0.07	0.06
like	0.29	0.06
foreign	0.04	0.15
films	0.08	0.11

What class will Naive bayes assign to the sentence "I always like foreign films."?

- 4.2** Given the following short movie reviews, each labeled with a genre, either comedy or action:

1. fun, couple, love, love **comedy**
2. fast, furious, shoot **action**
3. couple, fly, fast, fun, fun **comedy**
4. furious, shoot, shoot, fun **action**
5. fly, fast, shoot, love **action**

and a new document D:

fast, couple, shoot, fly

compute the most likely class for D. Assume a naive Bayes classifier and use add-1 smoothing for the likelihoods.

- 4.3** Train two models, multinomial naive Bayes and binarized naive Bayes, both with add-1 smoothing, on the following document counts for key sentiment words, with positive or negative class assigned as noted.

doc	“good”	“poor”	“great”	(class)
d1.	3	0	3	pos
d2.	0	1	2	pos
d3.	1	3	0	neg
d4.	1	5	2	neg
d5.	0	2	0	neg

Use both naive Bayes models to assign a class (pos or neg) to this sentence:

A good, good plot and great characters, but poor acting.

Do the two models agree or disagree?

Logistic Regression

"And how do you know that these fine begonias are not of equal importance?"

Hercule Poirot, in Agatha Christie's *The Mysterious Affair at Styles*

Detective stories are as littered with clues as texts are with words. Yet for the poor reader it can be challenging to know how to weigh the author's clues in order to make the crucial classification task: deciding whodunnit.

logistic regression

In this chapter we introduce an algorithm that is admirably suited for discovering the link between features or cues and some particular outcome: **logistic regression**. Indeed, logistic regression is one of the most important analytic tools in the social and natural sciences. In natural language processing, logistic regression is the baseline supervised machine learning algorithm for classification, and also has a very close relationship with neural networks. As we will see in Chapter 7, a neural network can be viewed as a series of logistic regression classifiers stacked on top of each other. Thus the classification and machine learning techniques introduced here will play an important role throughout the book.

Logistic regression can be used to classify an observation into one of two classes (like 'positive sentiment' and 'negative sentiment'), or into one of many classes. Because the mathematics for the two-class case is simpler, we'll describe this special case of logistic regression first in the next few sections, and then briefly summarize the use of **multinomial logistic regression** for more than two classes in Section 5.6.

We'll introduce the mathematics of logistic regression in the next few sections. But let's begin with some high-level issues.

Generative and Discriminative Classifiers: The most important difference between naive Bayes and logistic regression is that logistic regression is a **discriminative** classifier while naive Bayes is a **generative** classifier.

These are two very different frameworks for how to build a machine learning model. Consider a visual metaphor: imagine we're trying to distinguish dog images from cat images. A generative model would have the goal of understanding what dogs look like and what cats look like. You might literally ask such a model to 'generate', i.e. draw, a dog. Given a test image, the system then asks whether it's the cat model or the dog model that better fits (is less surprised by) the image, and chooses that as its label.



A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them). So maybe all the dogs in the training data are wearing collars and the cats aren't. If that one feature neatly separates the classes, the model is satisfied. If you ask such a model what it knows about cats all it can say is that they don't wear collars.

More formally, recall that the naive Bayes assigns a class c to a document d not by directly computing $P(c|d)$ but by computing a likelihood and a prior

$$\hat{c} = \operatorname{argmax}_{c \in C} P(d|c) P(c) \quad (5.1)$$

generative model
discriminative model

A **generative model** like naive Bayes makes use of this **likelihood** term, which expresses how to generate the features of a document *if we knew it was of class c*.

By contrast a **discriminative model** in this text categorization scenario attempts to **directly** compute $P(c|d)$. Perhaps it will learn to assign a high weight to document features that directly improve its ability to *discriminate* between possible classes, even if it couldn't generate an example of one of the classes.

Components of a probabilistic machine learning classifier: Like naive Bayes, logistic regression is a probabilistic classifier that makes use of supervised machine learning. Machine learning classifiers require a training corpus of M input/output pairs $(x^{(i)}, y^{(i)})$. (We'll use superscripts in parentheses to refer to individual instances in the training set—for sentiment classification each instance might be an individual document to be classified). A machine learning system for classification then has four components:

1. **A feature representation** of the input. For each input observation $x^{(i)}$, this will be a vector of features $[x_1, x_2, \dots, x_n]$. We will generally refer to feature i for input $x^{(j)}$ as $x_i^{(j)}$, sometimes simplified as x_i , but we will also see the notation f_i , $f_i(x)$, or, for multiclass classification, $f_i(c, x)$.
2. **A classification function** that computes \hat{y} , the estimated class, via $p(y|x)$. In the next section we will introduce the **sigmoid** and **softmax** tools for classification.
3. **An objective function** for learning, usually involving **minimizing error** on **training examples**. We will introduce the **cross-entropy loss function**.
4. **An algorithm** for optimizing the **objective function**. We introduce the **stochastic gradient descent** algorithm.

Logistic regression has two phases:

training: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.

test: Given a test example x we compute $p(y|x)$ and return the higher probability label $y = 1$ or $y = 0$.

5.1 Classification: the sigmoid

The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of a new input observation. Here we introduce the **sigmoid** classifier that will help us make this decision.

Consider a single input observation x , which we will represent by a vector of features $[x_1, x_2, \dots, x_n]$ (we'll show sample features in the next subsection). The classifier output y can be 1 (meaning the observation is a member of the class) or 0 (the observation is not a member of the class). We want to know the probability $P(y = 1|x)$ that this observation is a member of the class. So perhaps the decision

is “positive sentiment” versus “negative sentiment”, the features represent counts of words in a document, and $P(y = 1|x)$ is the probability that the document has positive sentiment, while $P(y = 0|x)$ is the probability that the document has negative sentiment.

Logistic regression solves this task by learning, from a training set, a vector of **weights** and a **bias term**. Each weight w_i is a real number, and is associated with one of the input features x_i . The weight w_i represents how important that input feature is to the classification decision, and can be positive (meaning the feature is associated with the class) or negative (meaning the feature is not associated with the class). Thus we might expect in a sentiment task the word *awesome* to have a high positive weight, and *abysmal* to have a very negative weight. The **bias term**, also called the **intercept**, is another real number that’s added to the weighted inputs.

To make a decision on a test instance—after we’ve learned the weights in training—the classifier first multiplies each x_i by its weight w_i , sums up the weighted features, and adds the bias term b . The resulting single number z expresses the weighted sum of the evidence for the class.

$$z = \left(\sum_{i=1}^n w_i x_i \right) + b \quad (5.2)$$

dot product

In the rest of the book we’ll represent such sums using the **dot product** notation from linear algebra. The dot product of two vectors a and b , written as $a \cdot b$ is the sum of the products of the corresponding elements of each vector. Thus the following is an equivalent formation to Eq. 5.2:

$$z = w \cdot x + b \quad (5.3)$$

But note that nothing in Eq. 5.3 forces z to be a legal probability, that is, to lie between 0 and 1. In fact, since weights are real-valued, the output might even be negative; z ranges from $-\infty$ to ∞ .

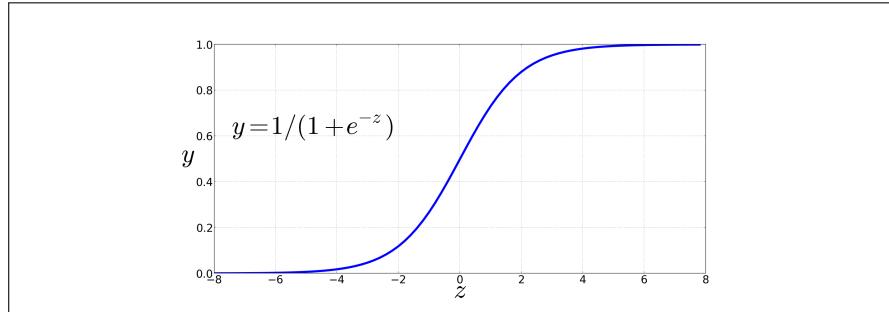


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

sigmoid logistic function

To create a probability, we’ll pass z through the **sigmoid** function, $\sigma(z)$. The sigmoid function (named because it looks like an s) is also called the **logistic function**, and gives logistic regression its name. The sigmoid has the following equation, shown graphically in Fig. 5.1:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (5.4)$$

The sigmoid has a number of advantages; it takes a real-valued number and maps it into the range $[0, 1]$, which is just what we want for a probability. Because it is

nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1. And it's differentiable, which as we'll see in Section 5.8 will be handy for learning.

We're almost there. If we apply the sigmoid to the sum of the weighted features, we get a number between 0 and 1. To make it a probability, we just need to make sure that the two cases, $p(y = 1)$ and $p(y = 0)$, sum to 1. We can do this as follows:

$$\begin{aligned} P(y = 1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1 + e^{-(w \cdot x + b)}} \\ P(y = 0) &= 1 - \sigma(w \cdot x + b) \\ &= 1 - \frac{1}{1 + e^{-(w \cdot x + b)}} \\ &= \frac{e^{-(w \cdot x + b)}}{1 + e^{-(w \cdot x + b)}} \end{aligned} \tag{5.5}$$

Now we have an algorithm that given an instance x computes the probability $P(y = 1|x)$. How do we make a decision? For a test instance x , we say yes if the probability $P(y = 1|x)$ is more than .5, and no otherwise. We call .5 the **decision boundary**:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

5.1.1 Example: sentiment classification

Let's have an example. Suppose we are doing binary sentiment classification on movie review text, and we would like to know whether to assign the sentiment class + or - to a review document doc . We'll represent each input observation by the 6 features $x_1 \dots x_6$ of the input shown in the following table; Fig. 5.2 shows the features in a sample mini test document.

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in doc$)	3
x_2	count(negative lexicon) $\in doc$)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in doc \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in doc$)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in doc \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\log(\text{word count of } doc)$	$\ln(66) = 4.19$

Let's assume for the moment that we've already learned a real-valued weight for each of these features, and that the 6 weights corresponding to the 6 features are $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$, while $b = 0.1$. (We'll discuss in the next section how the weights are learned.) The weight w_1 , for example indicates how important a feature the number of positive lexicon words (*great*, *nice*, *enjoyable*, etc.) is to a positive sentiment decision, while w_2 tells us the importance of negative lexicon words. Note that $w_1 = 2.5$ is positive, while $w_2 = -5.0$, meaning that negative words are negatively associated with a positive sentiment decision, and are about twice as important as positive words.

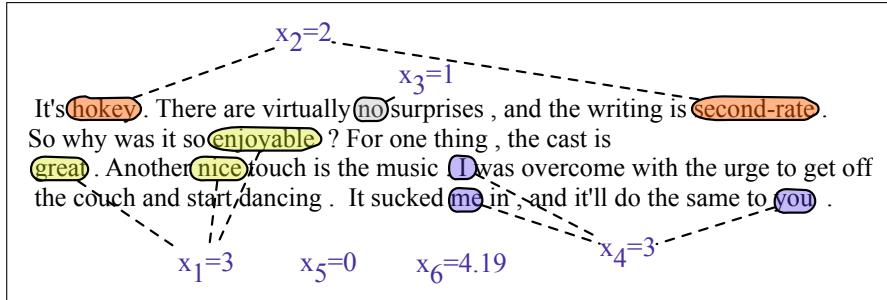


Figure 5.2 A sample mini test document showing the extracted features in the vector x .

Given these 6 features and the input review x , $P(+|x)$ and $P(-|x)$ can be computed using Eq. 5.5:

$$\begin{aligned} p(+|x) &= P(Y = 1|x) = \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned} \tag{5.6}$$

$$\begin{aligned} p(-|x) &= P(Y = 0|x) = 1 - \sigma(w \cdot x + b) \\ &= 0.30 \end{aligned}$$

Logistic regression is commonly applied to all sorts of NLP tasks, and any property of the input can be a feature. Consider the task of **period disambiguation**: deciding if a period is the end of a sentence or part of a word, by classifying each period into one of two classes EOS (end-of-sentence) and not-EOS. We might use features like x_1 below expressing that the current word is lower case and the class is EOS (perhaps with a positive weight), or that the current word is in our abbreviations dictionary (“Prof.”) and the class is EOS (perhaps with a negative weight). A feature can also express a quite complex combination of properties. For example a period following an upper case word is likely to be an EOS, but if the word itself is *St.* and the previous word is capitalized, then the period is likely part of a shortening of the word *street*.

$$\begin{aligned} x_1 &= \begin{cases} 1 & \text{if } \text{“Case}(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases} \\ x_2 &= \begin{cases} 1 & \text{if } w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases} \\ x_3 &= \begin{cases} 1 & \text{if } “w_i = \text{St.} \& \text{Case}(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Designing features: Features are generally designed by examining the training set with an eye to linguistic intuitions and the linguistic literature on the domain. A careful error analysis on the training set or devset of an early version of a system often provides insights into features.

For some tasks it is especially helpful to build complex features that are combinations of more primitive features. We saw such a feature for period disambiguation above, where a period on the word *St.* was less likely to be the end of the sentence if the previous word was capitalized. For logistic regression and naive Bayes these combination features or **feature interactions** have to be designed by hand.

feature
templates

For many tasks (especially when feature values can reference specific words) we'll need large numbers of features. Often these are created automatically via **feature templates**, abstract specifications of features. For example a bigram template for period disambiguation might create a feature for every pair of words that occurs before a period in the training set. Thus the feature space is sparse, since we only have to create a feature if that n-gram exists in that position in the training set. The feature is generally created as a hash from the string descriptions. A user description of a feature as, "bigram(American breakfast)" is hashed into a unique integer i that becomes the feature number f_i .

In order to avoid the extensive human effort of feature design, recent research in NLP has focused on **representation learning**: ways to learn features automatically in an unsupervised way from the input. We'll introduce methods for representation learning in Chapter 6 and Chapter 7.

Choosing a classifier Logistic regression has a number of advantages over naive Bayes. Naive Bayes has overly strong conditional independence assumptions. Consider two features which are strongly correlated; in fact, imagine that we just add the same feature f_1 twice. Naive Bayes will treat both copies of f_1 as if they were separate, multiplying them both in, overestimating the evidence. By contrast, logistic regression is much more robust to correlated features; if two features f_1 and f_2 are perfectly correlated, regression will simply assign part of the weight to w_1 and part to w_2 . Thus when there are many correlated features, logistic regression will assign a more accurate probability than naive Bayes. So logistic regression generally works better on larger documents or datasets and is a common default.

Despite the less accurate probabilities, naive Bayes still often makes the correct classification decision. Furthermore, naive Bayes can work extremely well (sometimes even better than logistic regression) on very small datasets (Ng and Jordan, 2002) or short documents (Wang and Manning, 2012). Furthermore, naive Bayes is easy to implement and very fast to train (there's no optimization step). So it's still a reasonable approach to use in some situations.

5.2 Learning in Logistic Regression

How are the parameters of the model, the weights w and bias b , learned? Logistic regression is an instance of supervised classification in which we know the correct label y (either 0 or 1) for each observation x . What the system produces via Eq. 5.5 is \hat{y} , the system's estimate of the true y . We want to learn parameters (meaning w and b) that make \hat{y} for each training observation as close as possible to the true y .

This requires 2 components that we foreshadowed in the introduction to the chapter. The first is a metric for how close the current label (\hat{y}) is to the true gold label y . Rather than measure similarity, we usually talk about the opposite of this: the *distance* between the system output and the gold output, and we call this distance the **loss function** or the **cost function**. In the next section we'll introduce the loss function that is commonly used for logistic regression and also for neural networks, the **cross-entropy loss**.

The second thing we need is an optimization algorithm for iteratively updating the weights so as to minimize this loss function. The standard algorithm for this is **gradient descent**; we'll introduce the **stochastic gradient descent** algorithm in the following section.

5.3 The cross-entropy loss function

We need a loss function that expresses, for an observation x , how close the classifier output ($\hat{y} = \sigma(w \cdot x + b)$) is to the correct output (y , which is 0 or 1). We'll call this:

$$L(\hat{y}, y) = \text{How much } \hat{y} \text{ differs from the true } y \quad (5.7)$$

cross-entropy loss

We do this via a loss function that prefers the correct class labels of the training examples to be *more likely*. This is called **conditional maximum likelihood estimation**: we choose the parameters w, b that **maximize the log probability of the true y labels in the training data** given the observations x . The resulting loss function is the negative log likelihood loss, generally called the **cross-entropy loss**.

Let's derive this loss function, applied to a single observation x . We'd like to learn weights that maximize the probability of the correct label $p(y|x)$. Since there are only two discrete outcomes (1 or 0), this is a Bernoulli distribution, and we can express the probability $p(y|x)$ that our classifier produces for one observation as the following (keeping in mind that if $y=1$, Eq. 5.8 simplifies to \hat{y} ; if $y=0$, Eq. 5.8 simplifies to $1 - \hat{y}$):

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y} \quad (5.8)$$

Now we take the log of both sides. This will turn out to be handy mathematically, and doesn't hurt us; whatever values maximize a probability will also maximize the log of the probability:

$$\begin{aligned} \log p(y|x) &= \log [\hat{y}^y (1 - \hat{y})^{1-y}] \\ &= y \log \hat{y} + (1 - y) \log (1 - \hat{y}) \end{aligned} \quad (5.9)$$

Eq. 5.9 describes a log likelihood that should be maximized. In order to turn this into loss function (something that we need to minimize), we'll just flip the sign on Eq. 5.9. The result is the cross-entropy loss L_{CE} :

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1 - y) \log (1 - \hat{y})] \quad (5.10)$$

Finally, we can plug in the definition of $\hat{y} = \sigma(w \cdot x + b)$:

$$L_{CE}(w, b) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \quad (5.11)$$

Let's see if this loss function does the right thing for our example from Fig. 5.2. We want the loss to be smaller if the model's estimate is close to correct, and bigger if the model is confused. So first let's suppose the correct gold label for the sentiment example in Fig. 5.2 is positive, i.e., $y = 1$. In this case our model is doing well, since from Eq. 5.6 it indeed gave the example a higher probability of being positive (.69) than negative (.31). If we plug $\sigma(w \cdot x + b) = .69$ and $y = 1$ into Eq. 5.11, the right side of the equation drops out, leading to the following loss:

$$\begin{aligned} L_{CE}(w, b) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log (.69) \\ &= .37 \end{aligned}$$

By contrast, let's pretend instead that the example in Fig. 5.2 was actually negative, i.e. $y = 0$ (perhaps the reviewer went on to say “But bottom line, the movie is terrible! I beg you not to see it!”). In this case our model is confused and we'd want the loss to be higher. Now if we plug $y = 0$ and $1 - \sigma(w \cdot x + b) = .31$ from Eq. 5.6 into Eq. 5.11, the left side of the equation drops out:

$$\begin{aligned} L_{CE}(w, b) &= -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \\ &= -[\log(1 - \sigma(w \cdot x + b))] \\ &= -\log(.31) \\ &= 1.17 \end{aligned}$$

Sure enough, the loss for the first classifier (.37) is less than the loss for the second classifier (1.17).

Why does minimizing this negative log probability do what we want? A perfect classifier would assign probability 1 to the correct outcome ($y=1$ or $y=0$) and probability 0 to the incorrect outcome. That means the higher \hat{y} (the closer it is to 1), the better the classifier; the lower \hat{y} is (the closer it is to 0), the worse the classifier. The negative log of this probability is a convenient loss metric since it goes from 0 (negative log of 1, no loss) to infinity (negative log of 0, infinite loss). This loss function also ensures that as the probability of the correct answer is maximized, the probability of the incorrect answer is minimized; since the two sum to one, any increase in the probability of the correct answer is coming at the expense of the incorrect answer. It's called the cross-entropy loss, because Eq. 5.9 is also the formula for the **cross-entropy** between the true probability distribution y and our estimated distribution \hat{y} .

Now we know what we want to minimize; in the next section, we'll see how to find the minimum.

5.4 Gradient Descent

Our goal with gradient descent is to find the optimal weights: minimize the loss function we've defined for the model. In Eq. 5.12 below, we'll explicitly represent the fact that the loss function L is parameterized by the weights, which we'll refer to in machine learning in general as θ (in the case of logistic regression $\theta = w, b$):

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^m L_{CE}(y^{(i)}, x^{(i)}; \theta) \quad (5.12)$$

How shall we find the minimum of this (or any) loss function? Gradient descent is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters θ) the function's slope is rising the most steeply, and moving in the opposite direction. The intuition is that if you are hiking in a canyon and trying to descend most quickly down to the river at the bottom, you might look around yourself 360 degrees, find the direction where the ground is sloping the steepest, and walk downhill in that direction.

convex

For logistic regression, this loss function is conveniently **convex**. A convex function has just one minimum; there are no local minima to get stuck in, so gradient descent starting from any point is guaranteed to find the minimum. (By contrast,

the loss for multi-layer neural networks is non-convex, and gradient descent may get stuck in local minima for neural network training and never find the global optimum.)

Although the algorithm (and the concept of gradient) are designed for direction vectors, let's first consider a visualization of the case where the parameter of our system is just a single scalar w , shown in Fig. 5.3.

Given a random initialization of w at some value w^1 , and assuming the loss function L happened to have the shape in Fig. 5.3, we need the algorithm to tell us whether at the next iteration we should move left (making w^2 smaller than w^1) or right (making w^2 bigger than w^1) to reach the minimum.

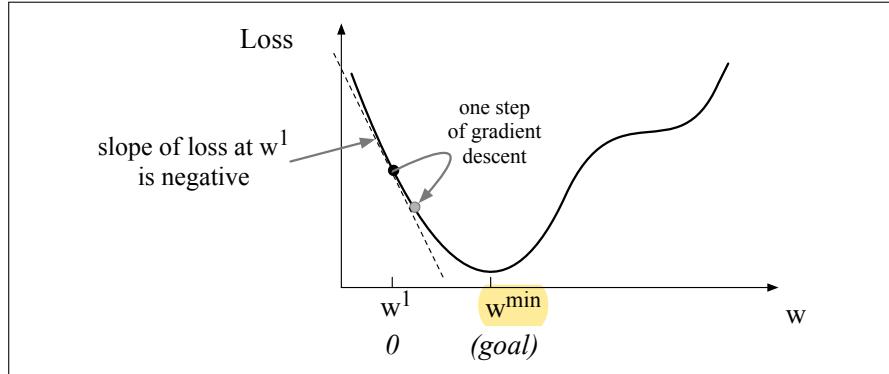


Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

gradient

The gradient descent algorithm answers this question by finding the **gradient** of the loss function at the current point and moving in the opposite direction. The gradient of a function of many variables is a vector pointing in the direction of the greatest increase in a function. The gradient is a multi-variable generalization of the slope, so for a function of one variable like the one in Fig. 5.3, we can informally think of the gradient as the slope. The dotted line in Fig. 5.3 shows the slope of this hypothetical loss function at point $w = w^1$. You can see that the slope of this dotted line is negative. Thus to find the minimum, gradient descent tells us to go in the opposite direction: moving w in a positive direction.

learning rate

The magnitude of the amount to move in gradient descent is the value of the slope $\frac{d}{dw}f(x; w)$ weighted by a **learning rate** η . A higher (faster) learning rate means that we should move w more on each step. The change we make in our parameter is the learning rate times the gradient (or the slope, in our single-variable example):

$$w^{t+1} = w^t - \eta \frac{d}{dw}f(x; w) \quad (5.13)$$

Now let's extend the intuition from a function of one scalar variable w to many variables, because we don't just want to move left or right, we want to know where in the N -dimensional space (of the N parameters that make up θ) we should move. The **gradient** is just such a vector; it expresses the directional components of the sharpest slope along each of those N dimensions. If we're just imagining two weight dimensions (say for one weight w and one bias b), the gradient might be a vector with two orthogonal components, each of which tells us how much the ground slopes in the w dimension and in the b dimension. Fig. 5.4 shows a visualization:

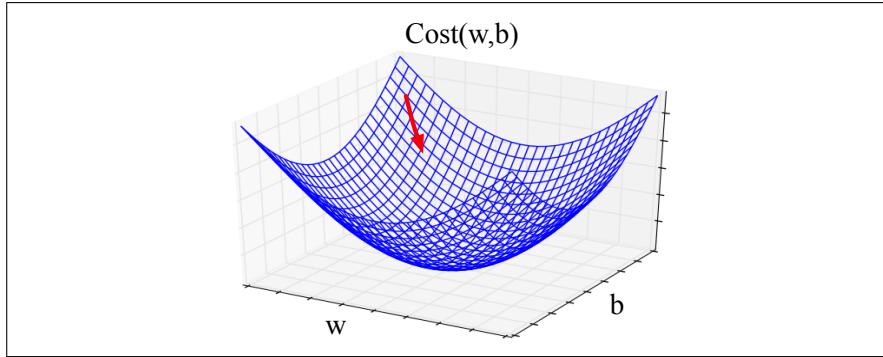


Figure 5.4 Visualization of the gradient vector in two dimensions w and b .

In an actual logistic regression, the parameter vector w is much longer than 1 or 2, since the input feature vector x can be quite long, and we need a weight w_i for each x_i . For each dimension/variable w_i in w (plus the bias b), the gradient will have a component that tells us the slope with respect to that variable. Essentially we're asking: "How much would a small change in that variable w_i influence the total loss function L ?"

In each dimension w_i , we express the slope as a partial derivative $\frac{\partial}{\partial w_i}$ of the loss function. The gradient is then defined as a vector of these partials. We'll represent \hat{y} as $f(x; \theta)$ to make the dependence on θ more obvious:

$$\nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(x; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x; \theta), y) \end{bmatrix} \quad (5.14)$$

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x; \theta), y) \quad (5.15)$$

5.4.1 The Gradient for Logistic Regression

In order to update θ , we need a definition for the gradient $\nabla L(f(x; \theta), y)$. Recall that for logistic regression, the cross-entropy loss function is:

$$L_{CE}(w, b) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))] \quad (5.16)$$

It turns out that the derivative of this function for one observation vector x is Eq. 5.17 (the interested reader can see Section 5.8 for the derivation of this equation):

$$\frac{\partial L_{CE}(w, b)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j \quad (5.17)$$

Note in Eq. 5.17 that the gradient with respect to a single weight w_j represents a very intuitive value: the difference between the true y and our estimated $\hat{y} = \sigma(w \cdot x + b)$ for that observation, multiplied by the corresponding input value x_j .

5.4.2 The Stochastic Gradient Descent Algorithm

Stochastic gradient descent is an online algorithm that minimizes the loss function by computing its gradient after each training example, and nudging θ in the right direction (the opposite direction of the gradient). Fig. 5.5 shows the algorithm.

```

function STOCHASTIC GRADIENT DESCENT( $L()$ ,  $f()$ ,  $x$ ,  $y$ ) returns  $\theta$ 
    # where: L is the loss function
    #     f is a function parameterized by  $\theta$ 
    #     x is the set of training inputs  $x^{(1)}$ ,  $x^{(2)}$ , ...,  $x^{(n)}$ 
    #     y is the set of training outputs (labels)  $y^{(1)}$ ,  $y^{(2)}$ , ...,  $y^{(n)}$ 

     $\theta \leftarrow 0$ 
    repeat til done # see caption
        For each training tuple  $(x^{(i)}, y^{(i)})$  (in random order)
            1. Optional (for reporting): # How are we doing on this tuple?
                Compute  $\hat{y}^{(i)} = f(x^{(i)}; \theta)$  # What is our estimated output  $\hat{y}$ ?
                Compute the loss  $L(\hat{y}^{(i)}, y^{(i)})$  # How far off is  $\hat{y}^{(i)}$  from the true output  $y^{(i)}$ ?
            2.  $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$  # How should we move  $\theta$  to maximize loss?
            3.  $\theta \leftarrow \theta - \eta g$  # Go the other way instead
        return  $\theta$ 

```

Figure 5.5 The stochastic gradient descent algorithm. Step 1 (computing the loss) is used to report how well we are doing on the current tuple. The algorithm can terminate when it converges (or when the gradient $< \epsilon$), or when progress halts (for example when the loss starts going up on a held-out set).

The learning rate η is a parameter that must be adjusted. If it's too high, the learner will take steps that are too large, overshooting the minimum of the loss function. If it's too low, the learner will take steps that are too small, and take too long to get to the minimum. It is common to begin the learning rate at a higher value, and then slowly decrease it, so that it is a function of the iteration k of training; you will sometimes see the notation η_k to mean the value of the learning rate at iteration k .

5.4.3 Working through an example

Let's walk though a single step of the gradient descent algorithm. We'll use a simplified version of the example in Fig. 5.2 as it sees a single observation x , whose correct value is $y = 1$ (this is a positive review), and with only two features:

$$\begin{aligned} x_1 &= 3 && \text{(count of positive lexicon words)} \\ x_2 &= 2 && \text{(count of negative lexicon words)} \end{aligned}$$

Let's assume the initial weights and bias in θ^0 are all set to 0, and the initial learning rate η is 0.1:

$$\begin{aligned} w_1 &= w_2 = b = 0 \\ \eta &= 0.1 \end{aligned}$$

The single update step requires that we compute the gradient, multiplied by the learning rate

$$\theta^{t+1} = \theta^t - \eta \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$$

In our mini example there are three parameters, so the gradient vector has 3 dimensions, for w_1 , w_2 , and b . We can compute the first gradient as follows:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

So after one step of gradient descent, the weights have shifted to be: $w_1 = .15$, $w_2 = .1$, and $b = .05$.

Note that this observation x happened to be a positive example. We would expect that after seeing more negative examples with high counts of negative words, that the weight w_2 would shift to have a negative value.

5.4.4 Mini-batch training

Stochastic gradient descent is called stochastic because it chooses a single random example at a time, moving the weights so as to improve performance on that single example. That can result in very choppy movements, so it's common to compute the gradient over batches of training instances rather than a single instance.

batch training

For example in **batch training** we compute the gradient over the entire dataset. By seeing so many examples, batch training offers a superb estimate of which direction to move the weights, at the cost of spending a lot of time processing every single example in the training set to compute this perfect direction.

mini-batch

A compromise is **mini-batch** training: we train on a group of m examples (perhaps 512, or 1024) that is less than the whole dataset. (If m is the size of the dataset, then we are doing **batch** gradient descent; if $m = 1$, we are back to doing stochastic gradient descent). Mini-batch training also has the advantage of computational efficiency. The mini-batches can easily be vectorized, choosing the size of the mini-batch based on the computational resources. This allows us to process all the examples in one mini-batch in parallel and then accumulate the loss, something that's not possible with individual or batch training.

We just need to define mini-batch versions of the cross-entropy loss function we defined in Section 5.3 and the gradient in Section 5.4.1. Let's extend the cross-entropy loss for one example from Eq. 5.10 to mini-batches of size m . We'll continue to use the notation that $x^{(i)}$ and $y^{(i)}$ mean the i th training features and training label, respectively. We make the assumption that the training examples are independent:

$$\log p(\text{training labels}) = \log \prod_{i=1}^m p(y^{(i)}|x^{(i)}) \quad (5.18)$$

$$= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}) \quad (5.19)$$

$$= -\sum_{i=1}^m L_{CE}(\hat{y}^{(i)}, y^{(i)}) \quad (5.20)$$

Now the cost function for the mini-batch of m examples is the average loss for each example:

$$\begin{aligned} Cost(w, b) &= \frac{1}{m} \sum_{i=1}^m L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\ &= -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log (1 - \sigma(w \cdot x^{(i)} + b)) \end{aligned} \quad (5.21)$$

The mini-batch gradient is the average of the individual gradients from Eq. 5.17:

$$\frac{\partial Cost(w, b)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m [\sigma(w \cdot x^{(i)} + b) - y^{(i)}] x_j^{(i)} \quad (5.22)$$

5.5 Regularization

Numquam ponenda est pluralitas sine necessitate
 ‘Plurality should never be proposed unless needed’
 William of Occam

There is a problem with learning weights that make the model perfectly match the training data. If a feature is perfectly predictive of the outcome because it happens to only occur in one class, it will be assigned a very high weight. The weights for features will attempt to perfectly fit details of the training set, in fact too perfectly, modeling noisy factors that just accidentally correlate with the class. This problem is called **overfitting**. A good model should be able to **generalize** well from the training data to the unseen test set, but a model that overfits will have poor generalization.

overfitting
generalize
regularization

To avoid overfitting, a new **regularization** term $R(\theta)$ is added to the objective function in Eq. 5.12, resulting in the following objective for a batch of m examples (slightly rewritten from Eq. 5.12 to be maximizing log probability rather than minimizing loss, and removing the $\frac{1}{m}$ term which doesn’t affect the argmax):

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) - \alpha R(\theta) \quad (5.23)$$

L2
regularization

The new regularization term $R(\theta)$ is used to penalize large weights. Thus a setting of the weights that matches the training data perfectly—but uses many weights with high values to do so—will be penalized more than a setting that matches the data a little less well, but does so using smaller weights. There are two common ways to compute this regularization term $R(\theta)$. **L2 regularization** is a quadratic function of the weight values, named because it uses the (square of the) L2 norm of the weight values. The L2 norm, $\|\theta\|_2$, is the same as the **Euclidean distance** of the vector θ from the origin. If θ consists of n weights, then:

$$R(\theta) = \|\theta\|_2^2 = \sum_{j=1}^n \theta_j^2 \quad (5.24)$$

The L2 regularized objective function becomes:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left[\sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^n \theta_j^2 \quad (5.25)$$

L1 regularization

L1 regularization is a linear function of the weight values, named after the L1 norm $\|W\|_1$, the sum of the absolute values of the weights, or **Manhattan distance** (the Manhattan distance is the distance you'd have to walk between two points in a city with a street grid like New York):

$$R(\theta) = \|\theta\|_1 = \sum_{i=1}^n |\theta_i| \quad (5.26)$$

The L1 regularized objective function becomes:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \left[\sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) \right] - \alpha \sum_{j=1}^n |\theta_j| \quad (5.27)$$

lasso
ridge

These kinds of regularization come from statistics, where L1 regularization is called **lasso regression** (Tibshirani, 1996) and L2 regularization is called **ridge regression**, and both are commonly used in language processing. L2 regularization is easier to optimize because of its simple derivative (the derivative of θ^2 is just 2θ), while L1 regularization is more complex (the derivative of $|\theta|$ is non-continuous at zero). But where L2 prefers weight vectors with many small weights, L1 prefers sparse solutions with some larger weights but many more weights set to zero. Thus L1 regularization leads to much sparser weight vectors, that is, far fewer features.

Both L1 and L2 regularization have Bayesian interpretations as constraints on the prior of how weights should look. L1 regularization can be viewed as a Laplace prior on the weights. L2 regularization corresponds to assuming that weights are distributed according to a gaussian distribution with mean $\mu = 0$. In a gaussian or normal distribution, the further away a value is from the mean, the lower its probability (scaled by the variance σ). By using a gaussian prior on the weights, we are saying that weights prefer to have the value 0. A gaussian for a weight θ_j is

$$\frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(\theta_j - \mu_j)^2}{2\sigma_j^2}\right) \quad (5.28)$$

If we multiply each weight by a gaussian prior on the weight, we are thus maximizing the following constraint:

$$\hat{\theta} = \operatorname{argmax}_{\theta} \prod_{i=1}^M P(y^{(i)} | x^{(i)}) \times \prod_{j=1}^n \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(\theta_j - \mu_j)^2}{2\sigma_j^2}\right) \quad (5.29)$$

which in log space, with $\mu = 0$, and assuming $2\sigma^2 = 1$, corresponds to

$$\hat{\theta} = \operatorname{argmax}_{\theta} \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^n \theta_j^2 \quad (5.30)$$

which is in the same form as Eq. 5.25.

5.6 Multinomial logistic regression

multinomial logistic regression

Sometimes we need more than two classes. Perhaps we might want to do 3-way sentiment classification (positive, negative, or neutral). Or we could be classifying the part of speech of a word (choosing from 10, 30, or even 50 different parts of speech), or assigning semantic labels like the named entities or semantic relations we will introduce in Chapter 18.

softmax

In such cases we use **multinomial logistic regression**, also called **softmax regression** (or, historically, the **maxent classifier**). In multinomial logistic regression the target y is a variable that ranges over more than two classes; we want to know the probability of y being in each potential class $c \in C$, $p(y = c|x)$.

The multinomial logistic classifier uses a generalization of the sigmoid, called the **softmax** function, to compute the probability $p(y = c|x)$. The softmax function takes a vector $z = [z_1, z_2, \dots, z_k]$ of k arbitrary values and maps them to a probability distribution, with each value in the range $(0,1)$, and all the values summing to 1. Like the sigmoid, it is an exponential function.

For a vector z of dimensionality k , the softmax is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \quad 1 \leq i \leq k \quad (5.31)$$

The softmax of an input vector $z = [z_1, z_2, \dots, z_k]$ is thus a vector itself:

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^k e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^k e^{z_i}}, \dots, \frac{e^{z_k}}{\sum_{i=1}^k e^{z_i}} \right] \quad (5.32)$$

The denominator $\sum_{i=1}^k e^{z_i}$ is used to normalize all the values into probabilities. Thus for example given a vector:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

the result softmax(z) is

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Again like the sigmoid, the input to the softmax will be the dot product between a weight vector w and an input vector x (plus a bias). But now we'll need separate weight vectors (and bias) for each of the K classes.

$$p(y = c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^k e^{w_j \cdot x + b_j}} \quad (5.33)$$

Like the sigmoid, the softmax has the property of squashing values toward 0 or 1. Thus if one of the inputs is larger than the others, it will tend to push its probability toward 1, and suppress the probabilities of the smaller inputs.

5.6.1 Features in Multinomial Logistic Regression

For multiclass classification the input features need to be a function of both the observation x and the candidate output class c . Thus instead of the notation x_i , f_i or $f_i(x)$, when we're discussing features we will use the notation $f_i(c, x)$, meaning feature i for a particular class c for a given observation x .

In binary classification, a positive weight on a feature pointed toward $y=1$ and a negative weight toward $y=0$, but in multiclass classification a feature could be evidence for or against an individual class.

Let's look at some sample features for a few NLP tasks to help understand this perhaps unintuitive use of features that are functions of both the observation x and the class c .

Suppose we are doing text classification, and instead of binary classification our task is to assign one of the 3 classes +, -, or 0 (neutral) to a document. Now a feature related to exclamation marks might have a negative weight for 0 documents, and a positive weight for + or - documents:

Var	Definition	Wt
$f_1(0, x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	-4.5
$f_1(+, x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	2.6
$f_1(-, x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1.3

5.6.2 Learning in Multinomial Logistic Regression

Multinomial logistic regression has a slightly different loss function than binary logistic regression because it uses the softmax rather than the sigmoid classifier. The loss function for a single example x is the sum of the logs of the K output classes:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\sum_{k=1}^K 1\{y=k\} \log p(y=k|x) \\ &= -\sum_{k=1}^K 1\{y=k\} \log \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^K e^{w_j \cdot x + b_j}} \end{aligned} \quad (5.34)$$

This makes use of the function $1\{\}$ which evaluates to 1 if the condition in the brackets is true and to 0 otherwise.

The gradient for a single example turns out to be very similar to the gradient for logistic regression, although we don't show the derivation here. It is the difference between the value for the true class k (which is 1) and the probability the classifier outputs for class k , weighted by the value of the input x_k :

$$\begin{aligned} \frac{\partial L_{CE}}{\partial w_k} &= -(1\{y=k\} - p(y=k|x))x_k \\ &= -\left(1\{y=k\} - \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^K e^{w_j \cdot x + b_j}}\right)x_k \end{aligned} \quad (5.35)$$

5.7 Interpreting models

Often we want to know more than just the correct classification of an observation. We want to know why the classifier made the decision it did. That is, we want our decision to be **interpretable**. Interpretability can be hard to define strictly, but the core idea is that as humans we should know why our algorithms reach the conclusions they do. Because the features to logistic regression are often human-designed, one way to understand a classifier's decision is to understand the role each feature plays in the decision. Logistic regression can be combined with statistical tests (the likelihood ratio test, or the Wald test); investigating whether a particular feature is significant by one of these tests, or inspecting its magnitude (how large is the weight w associated with the feature?) can help us interpret why the classifier made the decision it makes. This is enormously important for building transparent models.

Furthermore, in addition to its use as a classifier, logistic regression in NLP and many other fields is widely used as an analytic tool for testing hypotheses about the effect of various explanatory variables (features). In text classification, perhaps we want to know if logically negative words (*no*, *not*, *never*) are more likely to be associated with negative sentiment, or if negative reviews of movies are more likely to discuss the cinematography. However, in doing so it's necessary to control for potential confounds: other factors that might influence sentiment (the movie genre, the year it was made, perhaps the length of the review in words). Or we might be studying the relationship between NLP-extracted linguistic features and non-linguistic outcomes (hospital readmissions, political outcomes, or product sales), but need to control for confounds (the age of the patient, the county of voting, the brand of the product). In such cases, logistic regression allows us to test whether some feature is associated with some outcome above and beyond the effect of other features.

5.8 Advanced: Deriving the Gradient Equation

In this section we give the derivation of the gradient of the cross-entropy loss function L_{CE} for logistic regression. Let's start with some quick calculus refreshers. First, the derivative of $\ln(x)$:

$$\frac{d}{dx} \ln(x) = \frac{1}{x} \quad (5.36)$$

Second, the (very elegant) derivative of the sigmoid:

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)) \quad (5.37)$$

chain rule Finally, the **chain rule** of derivatives. Suppose we are computing the derivative of a composite function $f(x) = u(v(x))$. The derivative of $f(x)$ is the derivative of $u(x)$ with respect to $v(x)$ times the derivative of $v(x)$ with respect to x :

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx} \quad (5.38)$$

First, we want to know the derivative of the loss function with respect to a single weight w_j (we'll need to compute it for each weight, and for the bias):

$$\begin{aligned}
\frac{\partial LL(w,b)}{\partial w_j} &= \frac{\partial}{\partial w_j} - [y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\
&= - \left[\frac{\partial}{\partial w_j} y \log \sigma(w \cdot x + b) + \frac{\partial}{\partial w_j} (1-y) \log [1 - \sigma(w \cdot x + b)] \right]
\end{aligned} \tag{5.39}$$

Next, using the chain rule, and relying on the derivative of log:

$$\frac{\partial LL(w,b)}{\partial w_j} = - \frac{y}{\sigma(w \cdot x + b)} \frac{\partial}{\partial w_j} \sigma(w \cdot x + b) - \frac{1-y}{1-\sigma(w \cdot x + b)} \frac{\partial}{\partial w_j} 1 - \sigma(w \cdot x + b) \tag{5.40}$$

Rearranging terms:

$$\frac{\partial LL(w,b)}{\partial w_j} = - \left[\frac{y}{\sigma(w \cdot x + b)} - \frac{1-y}{1-\sigma(w \cdot x + b)} \right] \frac{\partial}{\partial w_j} \sigma(w \cdot x + b) \tag{5.41}$$

And now plugging in the derivative of the sigmoid, and using the chain rule one more time, we end up with Eq. 5.42:

$$\begin{aligned}
\frac{\partial LL(w,b)}{\partial w_j} &= - \left[\frac{y - \sigma(w \cdot x + b)}{\sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)]} \right] \sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)] \frac{\partial(w \cdot x + b)}{\partial w_j} \\
&= - \left[\frac{y - \sigma(w \cdot x + b)}{\sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)]} \right] \sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)] x_j \\
&= -[y - \sigma(w \cdot x + b)] x_j \\
&= [\sigma(w \cdot x + b) - y] x_j
\end{aligned} \tag{5.42}$$

5.9 Summary

This chapter introduced the **logistic regression** model of **classification**.

- Logistic regression is a supervised machine learning classifier that extracts real-valued features from the input, multiplies each by a weight, sums them, and passes the sum through a **sigmoid** function to generate a probability. A threshold is used to make a decision.
- Logistic regression can be used with two classes (e.g., positive and negative sentiment) or with multiple classes (**multinomial logistic regression**, for example for n-ary text classification, part-of-speech labeling, etc.).
- Multinomial logistic regression uses the **softmax** function to compute probabilities.
- The weights (vector w and bias b) are learned from a labeled training set via a loss function, such as the **cross-entropy loss**, that must be minimized.
- Minimizing this loss function is a **convex optimization** problem, and iterative algorithms like **gradient descent** are used to find the optimal weights.
- **Regularization** is used to avoid overfitting.
- Logistic regression is also one of the most useful analytic tools, because of its ability to transparently study the importance of individual features.

Bibliographical and Historical Notes

Logistic regression was developed in the field of statistics, where it was used for the analysis of binary data by the 1960s, and was particularly common in medicine ([Cox, 1969](#)). Starting in the late 1970s it became widely used in linguistics as one of the formal foundations of the study of linguistic variation ([Sankoff and Labov, 1979](#)).

Nonetheless, logistic regression didn't become common in natural language processing until the 1990s, when it seems to have appeared simultaneously from two directions. The first source was the neighboring fields of information retrieval and speech processing, both of which had made use of regression, and both of which lent many other statistical techniques to NLP. Indeed a very early use of logistic regression for document routing was one of the first NLP applications to use (LSI) embeddings as word representations ([Schütze et al., 1995](#)).

maximum entropy At the same time in the early 1990s logistic regression was developed and applied to NLP at IBM Research under the name **maximum entropy** modeling or **maxent** ([Berger et al., 1996](#)), seemingly independent of the statistical literature. Under that name it was applied to language modeling ([Rosenfeld, 1996](#)), part-of-speech tagging ([Ratnaparkhi, 1996](#)), parsing ([Ratnaparkhi, 1997](#)), coreference resolution ([Kehler, 1997b](#)), and text classification ([Nigam et al., 1999](#)).

More on classification can be found in machine learning textbooks ([Hastie et al. 2001, Witten and Frank 2005, Bishop 2006, Murphy 2012](#)).

Exercises

CHAPTER

6

Vector Semantics and Embeddings

The asphalt that Los Angeles is famous for occurs mainly on its freeways. But in the middle of the city is another patch of asphalt, the La Brea tar pits, and this asphalt preserves millions of fossil bones from the last of the Ice Ages of the Pleistocene Epoch. One of these fossils is the *Smilodon*, or sabre-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different sabre-tooth tiger called *Thylacosmilus* lived in Argentina and other parts of South America. *Thylacosmilus* was a marsupial whereas *Smilodon* was a placental mammal, but *Thylacosmilus* had the same long upper canines and, like *Smilodon*, had a protective bone flange on the lower jaw. The similarity of these two mammals is one of many examples of parallel or convergent evolution, in which particular contexts or environments lead to the evolution of very similar structures in different species (Gould, 1980).



The role of context is also important in the similarity of a less biological kind of organism: the word. Words that occur in *similar contexts* tend to have *similar meanings*. This link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis**. The hypothesis was first formulated in the 1950s by linguists like Joos (1950), Harris (1954), and Firth (1957), who noticed that words which are synonyms (like *oculist* and *eye-doctor*) tended to occur in the same environment (e.g., near words like *eye* or *examined*) with the amount of meaning difference between two words “corresponding roughly to the amount of difference in their environments” (Harris, 1954, 157).

In this chapter we introduce **vector semantics**, which instantiates this linguistic hypothesis by learning representations of the meaning of words, called **embeddings**, directly from their distributions in texts. These representations are used in every natural language processing application that makes use of meaning, and underlie the more powerful **contextualized word representations** like **ELMo** and **BERT** that we will introduce in Chapter 10.

These word representations are also the first example in this book of **representation learning**, automatically learning useful representations of the input text. Finding such **self-supervised** ways to learn representations of the input, instead of creating representations by hand via **feature engineering**, is an important focus of NLP research (Bengio et al., 2013).

We’ll begin, however, by introducing some basic principles of word meaning, which will motivate the vector semantic models of this chapter as well as extensions that we’ll return to in Chapter 19, Chapter 20, and Chapter 21.

distributional hypothesis

vector semantics embeddings

representation learning

6.1 Lexical Semantics

How should we represent the meaning of a word? In the N-gram models we saw in Chapter 3, and in many traditional NLP applications, our only representation of a word is as a string of letters, or perhaps as an index in a vocabulary list. This representation is not that different from a tradition in philosophy, perhaps you've seen it in introductory logic classes, in which the meaning of words is represented by just spelling the word with small capital letters; representing the meaning of "dog" as DOG, and "cat" as CAT).

Representing the meaning of a word by capitalizing it is a pretty unsatisfactory model. You might have seen the old philosophy joke:

Q: What's the meaning of life?

A: LIFE

Surely we can do better than this! After all, we'll want a model of word meaning to do all sorts of things for us. It should tell us that some words have similar meanings (*cat* is similar to *dog*), other words are antonyms (*cold* is the opposite of *hot*). It should know that some words have positive connotations (*happy*) while others have negative connotations (*sad*). It should represent the fact that the meanings of *buy*, *sell*, and *pay* offer differing perspectives on the same underlying purchasing event (If I buy something from you, you've probably sold it to me, and I likely paid you).

More generally, a model of word meaning should allow us to draw useful inferences that will help us solve meaning-related tasks like question-answering, summarization, detecting paraphrases or plagiarism, and dialogue.

lexical semantics

In this section we summarize some of these desiderata, drawing on results in the linguistic study of word meaning, which is called **lexical semantics**; we'll return to and expand on this list in Chapter 19.

lemma
citation form
wordform

Lemmas and Senses Let's start by looking at how one word (we'll choose *mouse*) might be defined in a dictionary:¹

- mouse (N)
 1. any of numerous small rodents...
 2. a hand-operated device that controls a cursor...

Here the form *mouse* is the **lemma**, also called the **citation form**. The form *mouse* would also be the lemma for the word *mice*; dictionaries don't have separate definitions for inflected forms like *mice*. Similarly *sing* is the lemma for *sing*, *sang*, *sung*. In many languages the infinitive form is used as the lemma for the verb, so Spanish *dormir* "to sleep" is the lemma for *duermes* "you sleep". The specific forms *sung* or *carpets* or *sing* or *duermes* are called **wordforms**.

As the example above shows, each lemma can have multiple meanings; the lemma *mouse* can refer to the rodent or the cursor control device. We call each of these aspects of the meaning of *mouse* a **word sense**. The fact that lemmas can be **polysemous** (have multiple senses) can make interpretation difficult (is someone who types "mouse info" into a search engine looking for a pet or a tool?). Chapter 19 will discuss the problem of polysemy, and introduce **word sense disambiguation**, the task of determining which sense of a word is being used in a particular context.

Synonymy One important component of word meaning is the relationship between word senses. For example when one word has a sense whose meaning is

¹ This example shortened from the online dictionary WordNet, discussed in Chapter 19.

synonym identical to a sense of another word, or nearly identical, we say the two senses of those two words are **synonyms**. Synonyms include such pairs as

couch/sofa vomit/throw up filbert/hazelnut car/automobile

propositional meaning

principle of contrast

A more formal definition of synonymy (between words rather than senses) is that two words are synonymous if they are substitutable one for the other in any sentence without changing the *truth conditions* of the sentence, the situations in which the sentence would be true. We often say in this case that the two words have the same **propositional meaning**.

While substitutions between some pairs of words like *car / automobile* or *water / H₂O* are truth preserving, the words are still not identical in meaning. Indeed, probably no two words are absolutely identical in meaning. One of the fundamental tenets of semantics, called the **principle of contrast** (Girard 1718, Bréal 1897, Clark 1987), is the assumption that a difference in linguistic form is always associated with at least some difference in meaning. For example, the word *H₂O* is used in scientific contexts and would be inappropriate in a hiking guide—*water* would be more appropriate—and this difference in genre is part of the meaning of the word. In practice, the word *synonym* is therefore commonly used to describe a relationship of approximate or rough synonymy.

similarity

Word Similarity While words don't have many synonyms, most words do have lots of *similar* words. *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly similar words. In moving from synonymy to similarity, it will be useful to shift from talking about relations between word senses (like synonymy) to relations between words (like similarity). Dealing with words avoids having to commit to a particular representation of word senses, which will turn out to simplify our task.

The notion of word **similarity** is very useful in larger semantic tasks. Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are, a very important component of natural language understanding tasks like question answering, paraphrasing, and summarization. One way of getting values for word similarity is to ask humans to judge how similar one word is to another. A number of datasets have resulted from such experiments. For example the SimLex-999 dataset (Hill et al., 2015) gives values on a scale from 0 to 10, like the examples below, which range from near-synonyms (*vanish, disappear*) to pairs that scarcely seem to have anything in common (*hole, agreement*):

vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

relatedness association

Word Relatedness The meaning of two words can be related in ways other than similarity. One such class of connections is called word **relatedness** (Budanitsky and Hirst, 2006), also traditionally called word **association** in psychology.

Consider the meanings of the words *coffee* and *cup*. Coffee is not similar to cup; they share practically no features (coffee is a plant or a beverage, while a cup is a manufactured object with a particular shape).

But coffee and cup are clearly related; they are associated by co-participating in an everyday event (the event of drinking coffee out of a cup). Similarly the nouns

scalpel and *surgeon* are not similar but are related eventively (a surgeon tends to make use of a scalpel).

semantic field

One common kind of relatedness between words is if they belong to the same **semantic field**. A semantic field is a set of words which cover a particular semantic domain and bear structured relations with each other.

For example, words might be related by being in the semantic field of hospitals (*surgeon, scalpel, nurse, anesthetic, hospital*), restaurants (*waiter, menu, plate, food, chef*), or houses (*door, roof, kitchen, family, bed*).

topic models

Semantic fields are also related to **topic models**, like **Latent Dirichlet Allocation, LDA**, which apply unsupervised learning on large sets of texts to induce sets of associated words from text. Semantic fields and topic models are very useful tools for discovering topical structure in documents.

In Chapter 19 we'll introduce even more relations between senses, including **hyponymy** or **IS-A**, **antonymy** (opposite meaning) and **meronymy** (part-whole relations).

semantic frame

Semantic Frames and Roles Closely related to semantic fields is the idea of a **semantic frame**. A semantic frame is a set of words that denote perspectives or participants in a particular type of event. A commercial transaction, for example, is a kind of event in which one entity trades money to another entity in return for some good or service, after which the good changes hands or perhaps the service is performed. This event can be encoded lexically by using verbs like *buy* (the event from the perspective of the buyer), *sell* (from the perspective of the seller), *pay* (focusing on the monetary aspect), or nouns like *buyer*. Frames have semantic roles (like *buyer, seller, goods, money*), and words in a sentence can take on these roles.

Knowing that *buy* and *sell* have this relation makes it possible for a system to know that a sentence like *Sam bought the book from Ling* could be paraphrased as *Ling sold the book to Sam*, and that Sam has the role of the *buyer* in the frame and Ling the *seller*. Being able to recognize such paraphrases is important for question answering, and can help in shifting perspective for machine translation.

connotations

sentiment

Connotation Finally, words have *affective meanings* or **connotations**. The word *connotation* has different meanings in different fields, but here we use it to mean the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations. For example some words have positive connotations (*happy*) while others have negative connotations (*sad*). Some words describe positive evaluation (*great, love*) and others negative evaluation (*terrible, hate*). Positive or negative evaluation expressed through language is called **sentiment**, as we saw in Chapter 4, and word sentiment plays a role in important tasks like sentiment analysis, stance detection, and many applications of natural language processing to the language of politics and consumer reviews.

Early work on affective meaning (Osgood et al., 1957) found that words varied along three important dimensions of affective meaning. These are now generally called *valence, arousal*, and *dominance*, defined as follows:

valence: the pleasantness of the stimulus

arousal: the intensity of emotion provoked by the stimulus

dominance: the degree of control exerted by the stimulus

Thus words like *happy* or *satisfied* are high on valence, while *unhappy* or *annoyed* are low on valence. *Excited* or *frenzied* are high on arousal, while *relaxed* or *calm* are low on arousal. *Important* or *controlling* are high on dominance, while *awed* or *influenced* are low on dominance. Each word is thus represented by three

numbers, corresponding to its value on each of the three dimensions, like the examples below:

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

Osgood et al. (1957) noticed that in using these 3 numbers to represent the meaning of a word, the model was representing each word as a point in a three-dimensional space, a vector whose three dimensions corresponded to the word's rating on the three scales. This revolutionary idea that word meaning word could be represented as a point in space (e.g., that part of the meaning of *heartbreak* can be represented as the point [2.45, 5.65, 3.58]) was the first expression of the vector semantics models that we introduce next.

6.2 Vector Semantics

How can we build a computational model that successfully deals with the different aspects of word meaning we saw in the previous section (word senses, word similarity and relatedness, lexical fields and frames, connotation)?

A perfect model that completely deals with each of these aspects of word meaning turns out to be elusive. But the current best model, called **vector semantics**, draws its inspiration from linguistic and philosophical work of the 1950's.

During that period, the philosopher Ludwig Wittgenstein, skeptical of the possibility of building a completely formal theory of meaning definitions for each word, suggested instead that “the meaning of a word is its use in the language” (Wittgenstein, 1953, PI 43). That is, instead of using some logical language to define each word, we should define words by some representation of how the word was used by actual people in speaking and understanding.

Linguists of the period like Joos (1950), Harris (1954), and Firth (1957) (the linguistic distributionalists), came up with a specific idea for realizing Wittgenstein's intuition: define a word by its environment or distribution in language use. A word's distribution is the set of contexts in which it occurs, the neighboring words or grammatical environments. The idea is that two words that occur in very similar distributions (that occur together with very similar words) are likely to have the same meaning.

Let's see an example illustrating this distributionalist approach. Suppose you didn't know what the Cantonese word *ongchoi* meant, but you do see it in the following sentences or contexts:

- (6.1) Ongchoi is delicious sauteed with garlic.
- (6.2) Ongchoi is superb over rice.
- (6.3) ...ongchoi leaves with salty sauces...

And furthermore let's suppose that you had seen many of these context words occurring in contexts like:

- (6.4) ...spinach sauteed with garlic over rice...

(6.5) ...chard stems and leaves are delicious...

(6.6) ...collard greens and other salty leafy greens

The fact that *ongchoi* occurs with words like *rice* and *garlic* and *delicious* and *salty*, as do words like *spinach*, *chard*, and *collard greens* might suggest to the reader that *ongchoi* is a leafy green similar to these other leafy greens.²

We can do the same thing computationally by just counting words in the context of *ongchoi*; we'll tend to see words like *sauteed* and *eaten* and *garlic*. The fact that these words and other similar context words also occur around the word *spinach* or *collard greens* can help us discover the similarity between these words and *ongchoi*.

Vector semantics thus combines two intuitions: the distributionalist intuition (defining a word by counting what other words occur in its environment), and the vector intuition of Osgood et al. (1957) we saw in the last section on connotation: defining the meaning of a word w as a vector, a list of numbers, a point in N -dimensional space. There are various versions of vector semantics, each defining the numbers in the vector somewhat differently, but in each case the numbers are based in some way on counts of neighboring words.

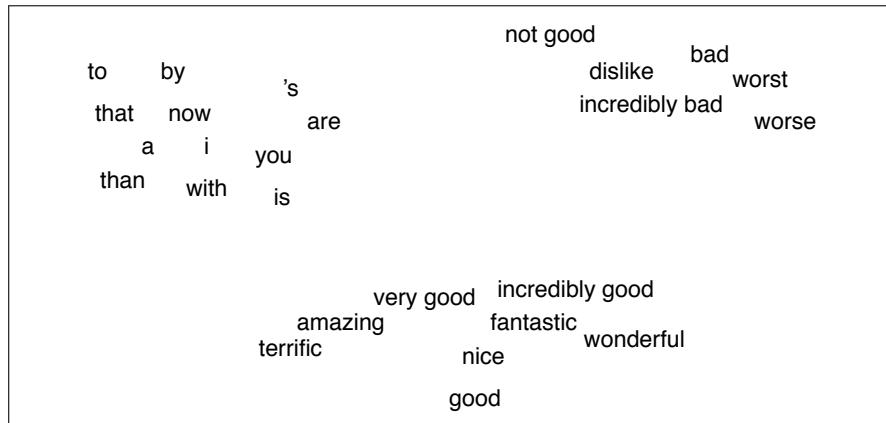


Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015).

The idea of vector semantics is thus to represent a word as a point in some multidimensional semantic space. Vectors for representing words are generally called **embeddings**, because the word is embedded in a particular vector space. Fig. 6.1 displays a visualization of embeddings that were learned for a sentiment analysis task, showing the location of some selected words projected down from the original 60-dimensional space into a two dimensional space.

Notice that positive and negative words seem to be located in distinct portions of the space (and different also from the neutral function words). This suggests one of the great advantages of vector semantics: it offers a fine-grained model of meaning that lets us also implement word similarity (and phrase similarity). For example, the sentiment analysis classifier we saw in Chapter 4 only works if enough of the important sentimental words that appear in the test set also appeared in the training set. But if words were represented as embeddings, we could assign sentiment as long as words with *similar meanings* as the test set words occurred in the training

² It's in fact *Ipomoea aquatica*, a relative of morning glory sometimes called *water spinach* in English.

set. Vector semantic models are also extremely practical because they can be learned automatically from text without any complex labeling or supervision.

As a result of these advantages, vector models of meaning are now the standard way to represent the meaning of words in NLP. In this chapter we'll introduce the two most commonly used models. First is the **tf-idf** model, often used as a baseline, in which the meaning of a word is defined by a simple function of the counts of nearby words. We will see that this method results in very long vectors that are **sparse**, i.e. contain mostly zeros (since most words simply never occur in the context of others).

Then we'll introduce the **word2vec** model, one of a family of models that are ways of constructing short, **dense** vectors that have useful semantic properties.

We'll also introduce the **cosine**, the standard way to use embeddings (vectors) to compute functions like *semantic similarity*, the similarity between two words, two sentences, or two documents, an important tool in practical applications like question answering, summarization, or automatic essay grading.

6.3 Words and Vectors

Vector or distributional models of meaning are generally based on a **co-occurrence matrix**, a way of representing how often words co-occur. This matrix can be constructed in various ways; let's begin by looking at one such co-occurrence matrix, a term-document matrix.

6.3.1 Vectors and documents

term-document matrix

In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents. Fig. 6.2 shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

vector space model

The term-document matrix of Fig. 6.2 was first defined as part of the **vector space model** of information retrieval (Salton, 1971). In this model, a document is represented as a count vector, a column in Fig. 6.3.

vector

To review some basic linear algebra, a **vector** is, at heart, just a list or array of numbers. So *As You Like It* is represented as the list [1,114,36,20] and *Julius Caesar* is represented as the list [7,62,1,2]. A **vector space** is a collection of vectors, characterized by their **dimension**. In the example in Fig. 6.3, the vectors are of dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality $|V|$, the vocabulary size.

vector space dimension

The ordering of the numbers in a vector space is not arbitrary; each position indicates a meaningful dimension on which the documents can vary. Thus the first dimension for both these vectors corresponds to the number of times the word *battle* occurs, and we can compare each dimension, noting for example that the vectors for *As You Like It* and *Twelfth Night* have similar values (1 and 0, respectively) for the first dimension.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

We can think of the vector for a document as identifying a point in $|V|$ -dimensional space; thus the documents in Fig. 6.3 are points in 4-dimensional space. Since 4-dimensional spaces are hard to draw in textbooks, Fig. 6.4 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.



Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

Term-document matrices were originally defined as a means of finding similar documents for the task of document **information retrieval**. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar. The vectors for the comedies *As You Like It* [1,114,36,20] and *Twelfth Night* [0,80,58,15] look a lot more like each other (more fools and wit than battles) than they look like *Julius Caesar* [7,62,1,2] or *Henry V* [13,89,4,3]. This is clear with the raw numbers; in the first dimension (battle) the comedies have low numbers and the others have high numbers, and we can see it visually in Fig. 6.4; we'll see very shortly how to quantify this intuition more formally.

A real term-document matrix, of course, wouldn't just have 4 rows and columns, let alone 2. More generally, the term-document matrix has $|V|$ rows (one for each word type in the vocabulary) and D columns (one for each document in the collection); as we'll see, vocabulary sizes are generally in the tens of thousands, and the number of documents can be enormous (think about all the pages on the web).

information retrieval

Information retrieval (IR) is the task of finding the document d from the D documents in some collection that best matches a query q . For IR we'll therefore also represent a query by a vector, also of length $|V|$, and we'll need a way to compare two vectors to find how similar they are. (Doing IR will also require efficient ways to store and manipulate these vectors by making use of the convenient fact that these vectors are sparse, i.e., mostly zeros).

Later in the chapter we'll introduce some of the components of this vector comparison process: the tf-idf term weighting, and the cosine similarity metric.

6.3.2 Words as vectors

We've seen that documents can be represented as vectors in a vector space. But vector semantics can also be used to represent the meaning of *words*, by associating each word with a vector.

row vector

The word vector is now a **row vector** rather than a column vector, and hence the dimensions of the vector are different. The four dimensions of the vector for *fool*, [36,58,1,4], correspond to the four Shakespeare plays. The same four dimensions are used to form the vectors for the other 3 words: *wit*, [20,15,2,3]; *battle*, [1,0,7,13]; and *good* [114,80,62,89]. Each entry in the vector thus represents the counts of the word's occurrence in the document corresponding to that dimension.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

word-word matrix

However, it is most common to use a different kind of context for the dimensions of a word's vector representation. Rather than the term-document matrix we use the **term-term matrix**, more commonly called the **word-word matrix** or the **term-context matrix**, in which the columns are labeled by words rather than documents. This matrix is thus of dimensionality $|V| \times |V|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, generally a window around the word, for example of 4 words to the left and 4 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a ± 4 word window around the row word. For example here is one example each of some words in their windows:

is traditionally followed by	cherry	pie, a traditional dessert
often mixed, such as	strawberry	rhubarb pie. Apple pie
computer peripherals and personal	digital	assistants. These devices usually
a computer. This includes	information	available on the internet

If we then take every occurrence of each word (say **strawberry**) and count the context words around it, we get a word-word co-occurrence matrix. Fig. 6.5 shows a simplified subset of the word-word co-occurrence matrix for these four words computed from the Wikipedia corpus (Davies, 2015).

Note in Fig. 6.5 that the two words *cherry* and *strawberry* are more similar to each other (both *pie* and *sugar* tend to occur in their window) than they are to other words like *digital*; conversely, *digital* and *information* are more similar to each other than, say, to *strawberry*. Fig. 6.6 shows a spatial visualization.

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	
strawberry	0	...	0	0	1	60	19	
digital	0	...	1670	1683	85	5	4	
information	0	...	3325	3982	378	5	13	

Figure 6.5 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.



Figure 6.6 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

Note that $|V|$, the length of the vector, is generally the size of the vocabulary, usually between 10,000 and 50,000 words (using the most frequent words in the training corpus; keeping words after about the most frequent 50,000 or so is generally not helpful). But of course since most of these numbers are zero these are **sparse** vector representations, and there are efficient algorithms for storing and computing with sparse matrices.

Now that we have some intuitions, let's move on to examine the details of computing word similarity. Afterwards we'll discuss the tf-idf method of weighting cells.

6.4 Cosine for measuring similarity

To define similarity between two target words v and w , we need a measure for taking two such vectors and giving a measure of vector similarity. By far the most common similarity metric is the **cosine** of the angle between the vectors.

The cosine—like most measures for vector similarity used in NLP—is based on the **dot product** operator from linear algebra, also called the **inner product**:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N \quad (6.7)$$

As we will see, most metrics for similarity between vectors are based on the dot product. The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong dissimilarity.

This raw dot product, however, has a problem as a similarity metric: it favors **vector length** **long** vectors. The **vector length** is defined as

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2} \quad (6.8)$$

The dot product is higher if a vector is longer, with higher values in each dimension. More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them. The raw dot product thus will be higher for frequent words. But this is a problem; we'd like a similarity metric that tells us how similar two words are regardless of their frequency.

The simplest way to modify the dot product to normalize for the vector length is to divide the dot product by the lengths of each of the two vectors. This **normalized dot product** turns out to be the same as the cosine of the angle between the two vectors, following from the definition of the dot product between two vectors \mathbf{a} and \mathbf{b} :

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}||\mathbf{b}|\cos\theta \\ \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} &= \cos\theta \end{aligned} \quad (6.9)$$

cosine The **cosine** similarity metric between two vectors \mathbf{v} and \mathbf{w} thus can be computed as:

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}} \quad (6.10)$$

For some applications we pre-normalize each vector, by dividing it by its length, creating a **unit vector** of length 1. Thus we could compute a unit vector from \mathbf{a} by dividing it by $|\mathbf{a}|$. For unit vectors, the dot product is the same as the cosine.

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for vectors that are orthogonal, to -1 for vectors pointing in opposite directions. But raw frequency values are non-negative, so the cosine for these vectors ranges from 0–1.

Let's see how the cosine computes which of the words *cherry* or *digital* is closer in meaning to *information*, just using raw counts from the following shortened table:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry}, \text{information}) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

The model decides that *information* is way closer to *digital* than it is to *cherry*, a result that seems sensible. Fig. 6.7 shows a visualization.

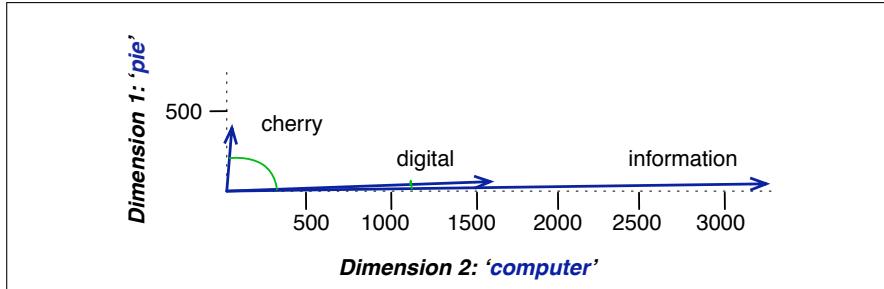


Figure 6.7 A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. Note that the angle between *digital* and *information* is smaller than the angle between *cherry* and *information*. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0°); the cosine of all other angles is less than 1.

6.5 TF-IDF: Weighing terms in the vector

The co-occurrence matrix in Fig. 6.5 represented each cell by the raw frequency of the co-occurrence of two words.

It turns out, however, that simple frequency isn't the best measure of association between words. One problem is that raw frequency is very skewed and not very discriminative. If we want to know what kinds of contexts are shared by *cherry* and *strawberry* but not by *digital* and *information*, we're not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words and aren't informative about any particular word. We saw this also in Fig. 6.3 for the Shakespeare corpus; the dimension for the word *good* is not very discriminative between plays; *good* is simply a frequent word and has roughly equivalent high frequencies in each of the plays.

It's a bit of a paradox. Words that occur nearby frequently (maybe *pie* nearby *cherry*) are more important than words that only appear once or twice. Yet words that are too frequent—ubiquitous, like *the* or *good*—are unimportant. How can we balance these two conflicting constraints?

The **tf-idf algorithm** (the ‘-’ here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions:

term frequency (Luhn, 1957): the frequency of the word t in the document d . We can just use the raw count as the term frequency:

$$\text{tf}_{t,d} = \text{count}(t,d) \quad (6.11)$$

Alternatively we can squash the raw frequency a bit, by using the \log_{10} of the frequency instead. The intuition is that a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document. Because we can't take the log of 0, we normally add 1 to the count:³

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d) + 1) \quad (6.12)$$

If we use log weighting, terms which occur 10 times in a document would have a $\text{tf}=2$, 100 times in a document $\text{tf}=3$, 1000 times $\text{tf}=4$, and so on.

³ Or we can use this alternative: $\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$

document frequency

The second factor is used to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful. The **document frequency** df_t of a term t is the number of documents it occurs in. Document frequency is not the same as the **collection frequency** of a term, which is the total number of times the word appears in the whole collection in any document. Consider in the collection of Shakespeare's 37 plays the two words *Romeo* and *action*. The words have identical collection frequencies (they both occur 113 times in all the plays) but very different document frequencies, since *Romeo* only occurs in a single play. If our goal is find documents about the romantic tribulations of Romeo, the word *Romeo* should be highly weighted, but not *action*:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

idf

We emphasize discriminative words like *Romeo* via the **inverse document frequency** or **idf** term weight (Sparck Jones, 1972). The idf is defined using the fraction N/df_t , where N is the total number of documents in the collection, and df_t is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. It's usually clear what counts as a document: in Shakespeare we would use a play; when processing a collection of encyclopedia articles like Wikipedia, the document is a Wikipedia page; in processing newspaper articles, the document is a single article. Occasionally your corpus might not have appropriate document divisions and you might need to break up the corpus into documents yourself for the purposes of computing idf.

Because of the large number of documents in many collections, this measure too is usually squashed with a log function. The resulting definition for inverse document frequency (idf) is thus

$$\text{idf}_t = \log_{10} \left(\frac{N}{df_t} \right) \quad (6.13)$$

Here are some idf values for some words in the Shakespeare corpus, ranging from extremely informative words which occur in only one play like *Romeo*, to those that occur in a few like *salad* or *Falstaff*, to those which are very common like *fool* or so common as to be completely non-discriminative since they occur in all 37 plays like *good* or *sweet*.⁴

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

⁴ *Sweet* was one of Shakespeare's favorite adjectives, a fact probably related to the increased use of sugar in European recipes around the turn of the 16th century (Jurafsky, 2014, p. 175).

tf-idf

The **tf-idf** weighted value $w_{t,d}$ for word t in document d thus combines term frequency $\text{tf}_{t,d}$ (defined either by Eq. 6.11 or by Eq. 6.12) with idf from Eq. 6.13:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \quad (6.14)$$

Fig. 6.8 applies tf-idf weighting to the Shakespeare term-document matrix in Fig. 6.2, using the tf equation Eq. 6.12. Note that the tf-idf values for the dimension corresponding to the word *good* have now all become 0; since this word appears in every document, the tf-idf algorithm leads it to be ignored in any comparison of the plays. Similarly, the word *fool*, which appears in 36 out of the 37 plays, has a much lower weight.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20 + 1) = 1.322$ and $\text{idf} = .037$. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

The tf-idf weighting is the way for weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing. It's also a great baseline, the simple thing to try first. We'll look at other weightings like PPMI (Positive Pointwise Mutual Information) in Section 6.7.

6.6 Applications of the tf-idf vector model

In summary, the vector semantics model we've described so far represents a target word as a vector with dimensions corresponding to all the words in the vocabulary (length $|V|$, with vocabularies of 20,000 to 50,000), which is also sparse (most values are zero). The values in each dimension are the frequency with which the target word co-occurs with each neighboring context word, weighted by tf-idf. The model computes the similarity between two words x and y by taking the cosine of their tf-idf vectors; high cosine, high similarity. This entire model is sometimes referred to for short as the **tf-idf** model, after the weighting function.

One common use for a tf-idf model is to compute word similarity, a useful tool for tasks like finding word paraphrases, tracking changes in word meaning, or automatically discovering meanings of words in different corpora. For example, we can find the 10 most similar words to any target word w by computing the cosines between w and each of the $V - 1$ other words, sorting, and looking at the top 10.

centroid**document vector**

The tf-idf vector model can also be used to decide if two documents are similar. We represent a document by taking the vectors of all the words in the document, and computing the **centroid** of all those vectors. The centroid is the multidimensional version of the mean; the centroid of a set of vectors is a single vector that has the minimum sum of squared distances to each of the vectors in the set. Given k word vectors w_1, w_2, \dots, w_k , the centroid **document vector** d is:

$$d = \frac{w_1 + w_2 + \dots + w_k}{k} \quad (6.15)$$

Given two documents, we can then compute their document vectors d_1 and d_2 , and estimate the similarity between the two documents by $\cos(d_1, d_2)$.

Document similarity is also useful for all sorts of applications; information retrieval, plagiarism detection, news recommender systems, and even for digital humanities tasks like comparing different versions of a text to see which are similar to each other.

6.7 Optional: Pointwise Mutual Information (PMI)

An alternative weighting function to tf-idf is called PPMI (positive pointwise mutual information). PPMI draws on the intuition that the best way to weigh the association between two words is to ask how much **more** the two words co-occur in our corpus than we would have a priori expected them to appear by chance.

pointwise mutual information

Pointwise mutual information (Fano, 1961)⁵ is one of the most important concepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} \quad (6.17)$$

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$\text{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)} \quad (6.18)$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated.

PPMI

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring *less often* than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each 10^{-6} occur together less often than chance, we would need to be certain that the probability of the two occurring together is significantly different than 10^{-12} , and this kind of granularity would require an enormous corpus. Furthermore it's not clear whether it's even possible to evaluate such scores of 'unrelatedness' with human judgments. For this reason it is more common to use Positive PMI (called **PPMI**) which replaces all negative PMI values

⁵ Pointwise mutual information is based on the **mutual information** between two random variables X and Y , which is defined as:

$$I(X, Y) = \sum_x \sum_y P(x, y) \log_2 \frac{P(x, y)}{P(x)P(y)} \quad (6.16)$$

In a confusion of terminology, Fano used the phrase *mutual information* to refer to what we now call *pointwise mutual information* and the phrase *expectation of the mutual information* for what we now call *mutual information*

with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994)⁶:

$$\text{PPMI}(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0\right) \quad (6.19)$$

More formally, let's assume we have a co-occurrence matrix F with W rows (words) and C columns (contexts), where f_{ij} gives the number of times word w_i occurs in context c_j . This can be turned into a PPMI matrix where $ppmi_{ij}$ gives the PPMI value of word w_i with context c_j as follows:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad p_{i*} = \frac{\sum_{j=1}^C f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad p_{*j} = \frac{\sum_{i=1}^W f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad (6.20)$$

$$\text{PPMI}_{ij} = \max\left(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0\right) \quad (6.21)$$

Let's see some PPMI calculations. We'll use Fig. 6.9, which repeats Fig. 6.5 plus all the count marginals, and let's pretend for ease of calculation that these are the only words/context that matter.

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Figure 6.9 Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other words/context matter.

Thus for example we could compute $\text{PPMI}(w=\text{information}, c=\text{data})$, assuming we pretended that Fig. 6.5 encompassed all the relevant word contexts/dimensions, as follows:

$$\begin{aligned} P(w=\text{information}, c=\text{data}) &= \frac{3982}{11716} = .3399 \\ P(w=\text{information}) &= \frac{7703}{11716} = .6575 \\ P(c=\text{data}) &= \frac{5673}{11716} = .4842 \\ \text{ppmi}(\text{information}, \text{data}) &= \log 2(.3399 / (.6575 * .4842)) = .0944 \end{aligned}$$

Fig. 6.10 shows the joint probabilities computed from the counts in Fig. 6.9, and Fig. 6.11 shows the PPMI values. Not surprisingly, *cherry* and *strawberry* are highly associated with both *pie* and *sugar*, and *data* is mildly associated with *information*.

PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values. One way to reduce this bias toward low frequency events is to slightly change the computation for $P(c)$, using a different function $P_\alpha(c)$ that raises the probability of the context word to the power of α :

$$\text{PPMI}_\alpha(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P_\alpha(c)}, 0\right) \quad (6.22)$$

⁶ Positive PMI also cleanly solves the problem of what to do with zero counts, using 0 to replace the $-\infty$ from $\log(0)$.

	$p(w, \text{context})$					$p(w)$
	computer	data	result	pie	sugar	$p(w)$
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
$p(\text{context})$	0.4265	0.4842	0.0404	0.0437	0.0052	

Figure 6.10 Replacing the counts in Fig. 6.5 with joint probabilities, showing the marginals around the outside.

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Figure 6.11 The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 6.10. Note that most of the 0 PPMI values are ones that had a negative PMI; for example $\text{PMI}(\text{cherry}, \text{computer}) = -6.7$, meaning that *cherry* and *computer* co-occur on Wikipedia less often than we would expect by chance, and with PPMI we replace negative values by zero.

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}(c)^\alpha} \quad (6.23)$$

Levy et al. (2015) found that a setting of $\alpha = 0.75$ improved performance of embeddings on a wide range of tasks (drawing on a similar weighting used for skip-grams described below in Eq. 6.32). This works because raising the count to $\alpha = 0.75$ increases the probability assigned to rare contexts, and hence lowers their PMI ($P_\alpha(c) > P(c)$ when c is rare).

Another possible solution is Laplace smoothing: Before computing PMI, a small constant k (values of 0.1-3 are common) is added to each of the counts, shrinking (discounting) all the non-zero values. The larger the k , the more the non-zero counts are discounted.

6.8 Word2vec

In the previous sections we saw how to represent a word as a sparse, long vector with dimensions corresponding to the words in the vocabulary, and whose values were tf-idf or PPMI functions of the count of the word co-occurring with each neighboring word. In this section we turn to an alternative method for representing a word: the use of vectors that are **short** (of length perhaps 50-1000) and **dense** (most values are non-zero).

It turns out that dense vectors work better in every NLP task than sparse vectors. While we don't completely understand all the reasons for this, we have some intuitions. First, dense vectors may be more successfully included as features in machine learning systems; for example if we use 100-dimensional word embeddings as features, a classifier can just learn 100 weights to represent a function of word meaning; if we instead put in a 50,000 dimensional vector, a classifier would have to learn tens of thousands of weights for each of the sparse dimensions. Second, because they contain fewer parameters than sparse vectors of explicit counts,

dense vectors may generalize better and help avoid overfitting. Finally, dense vectors may do a better job of capturing synonymy than sparse vectors. For example, *car* and *automobile* are synonyms; but in a typical sparse vector representation, the *car* dimension and the *automobile* dimension are distinct dimensions. Because the relationship between these two dimensions is not modeled, sparse vectors may fail to capture the similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor.

skip-gram
SGNS
word2vec

In this section we introduce one method for very dense, short vectors, **skip-gram with negative sampling**, sometimes called **SGNS**. The skip-gram algorithm is one of two algorithms in a software package called **word2vec**, and so sometimes the algorithm is loosely referred to as word2vec (Mikolov et al. 2013, Mikolov et al. 2013a). The word2vec methods are fast, efficient to train, and easily available online with code and pretrained embeddings. We point to other embedding methods, like the equally popular **GloVe** (Pennington et al., 2014), at the end of the chapter.

The intuition of word2vec is that instead of counting how often each word w occurs near, say, *apricot*, we'll instead train a classifier on a binary prediction task: “Is word w likely to show up near *apricot*?” We don't actually care about this prediction task; instead we'll take the learned classifier *weights* as the word embeddings.

The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier; a word s that occurs near the target word *apricot* acts as gold ‘correct answer’ to the question “Is word w likely to show up near *apricot*?” This avoids the need for any sort of hand-labeled supervision signal. This idea was first proposed in the task of neural language modeling, when Bengio et al. (2003) and Collobert et al. (2011) showed that a neural language model (a neural network that learned to predict the next word from prior words) could just use the next word in running text as its supervision signal, and could be used to learn an embedding representation for each word as part of doing this prediction task.

We'll see how to do neural networks in the next chapter, but word2vec is a much simpler model than the neural network language model, in two ways. First, word2vec simplifies the task (making it binary classification instead of word prediction). Second, word2vec simplifies the architecture (training a logistic regression classifier instead of a multi-layer neural network with hidden layers that demand more sophisticated training algorithms). The intuition of skip-gram is:

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the regression weights as the embeddings.

6.8.1 The classifier

Let's start by thinking about the classification task, and then turn to how to train. Imagine a sentence like the following, with a target word *apricot*, and assume we're using a window of ± 2 context words:

... lemon, a [tablespoon of apricot jam, a] pinch ...
c1 c2 t c3 c4

Our goal is to train a classifier such that, given a tuple (t, c) of a target word t paired with a candidate context word c (for example *(apricot, jam)*, or perhaps

(*apricot, aardvark*) it will return the probability that c is a real context word (true for *jam*, false for *aardvark*):

$$P(+|t, c) \quad (6.24)$$

The probability that word c is not a real context word for t is just 1 minus Eq. 6.24:

$$P(-|t, c) = 1 - P(+|t, c) \quad (6.25)$$

How does the classifier compute the probability P ? The intuition of the skip-gram model is to base this probability on similarity: a word is likely to occur near the target if its embedding is similar to the target embedding. How can we compute similarity between embeddings? Recall that two vectors are similar if they have a high dot product (cosine, the most popular similarity metric, is just a normalized dot product). In other words:

$$\text{Similarity}(t, c) \approx t \cdot c \quad (6.26)$$

Of course, the dot product $t \cdot c$ is not a probability, it's just a number ranging from $-\infty$ to ∞ . (Recall, for that matter, that cosine isn't a probability either). To turn the dot product into a probability, we'll use the **logistic** or **sigmoid** function $\sigma(x)$, the fundamental core of logistic regression:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (6.27)$$

The probability that word c is a real context word for target word t is thus computed as:

$$P(+|t, c) = \frac{1}{1 + e^{-t \cdot c}} \quad (6.28)$$

The sigmoid function just returns a number between 0 and 1, so to make it a probability we'll need to make sure that the total probability of the two possible events (c being a context word, and c not being a context word) sums to 1.

The probability that word c is not a real context word for t is thus:

$$\begin{aligned} P(-|t, c) &= 1 - P(+|t, c) \\ &= \frac{e^{-t \cdot c}}{1 + e^{-t \cdot c}} \end{aligned} \quad (6.29)$$

Equation 6.28 gives us the probability for one word, but we need to take account of the multiple context words in the window. Skip-gram makes the strong but very useful simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|t, c_{1:k}) = \prod_{i=1}^k \frac{1}{1 + e^{-t \cdot c_i}} \quad (6.30)$$

$$\log P(+|t, c_{1:k}) = \sum_{i=1}^k \log \frac{1}{1 + e^{-t \cdot c_i}} \quad (6.31)$$

In summary, skip-gram trains a probabilistic classifier that, given a test target word t and its context window of k words $c_{1:k}$, assigns a probability based on how similar

this context window is to the target word. The probability is based on applying the logistic (sigmoid) function to the dot product of the embeddings of the target word with each context word. We could thus compute this probability if only we had embeddings for each target word and context word in the vocabulary. Let's now turn to learning these embeddings (which is the real goal of training this classifier in the first place).

6.8.2 Learning skip-gram embeddings

Word2vec learns embeddings by starting with an initial set of embedding vectors and then iteratively shifting the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby. Let's start by considering a single piece of training data:

```
... lemon, a [tablespoon of apricot jam,      a] pinch ...
          c1        c2     t     c3      c4
```

This example has a target word t (apricot), and 4 context words in the $L = \pm 2$ window, resulting in 4 positive training instances (on the left below):

positive examples +		negative examples -	
t	c	t	c
apricot	tablespoon	apricot	aardvark
apricot	of	apricot	my
apricot	jam	apricot	where
apricot	a	apricot	coaxial
		apricot	seven
		apricot	forever
		apricot	dear
		apricot	if

For training a binary classifier we also need negative examples. In fact skip-gram uses more negative examples than positive examples (with the ratio between them set by a parameter k). So for each of these (t, c) training instances we'll create k negative samples, each consisting of the target t plus a 'noise word'. A noise word is a random word from the lexicon, constrained not to be the target word t . The right above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set – for each positive example t, c .

The noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight. If we were sampling according to unweighted frequency $p(w)$, it would mean that with unigram probability $p("the")$ we would choose the word *the* as a noise word, with unigram probability $p("aardvark")$ we would choose *aardvark*, and so on. But in practice it is common to set $\alpha = .75$, i.e. use the weighting $p^{\frac{3}{4}}(w)$:

$$P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha} \quad (6.32)$$

Setting $\alpha = .75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$. To visualize this intuition, it might help to work out the probabilities for an example with two events, $P(a) = .99$ and $P(b) = .01$:

$$\begin{aligned} P_\alpha(a) &= \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \\ P_\alpha(b) &= \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03 \end{aligned} \quad (6.33)$$

Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings such that we

- Maximize the similarity of the target word, context word pairs (t, c) drawn from the positive examples
- Minimize the similarity of the (t, c) pairs drawn from the negative examples.

We can express this formally over the whole training set as:

$$L(\theta) = \sum_{(t,c) \in +} \log P(+|t, c) + \sum_{(t,c) \in -} \log P(-|t, c) \quad (6.34)$$

If we look at one word/context pair (t, c) with its k noise words $n_1 \dots n_k$, the learning objective L is:

$$\begin{aligned} L(\theta) &= \log P(+|t, c) + \sum_{i=1}^k \log P(-|t, n_i) \\ &= \log \sigma(c \cdot t) + \sum_{i=1}^k \log \sigma(-n_i \cdot t) \\ &= \log \frac{1}{1 + e^{-c \cdot t}} + \sum_{i=1}^k \log \frac{1}{1 + e^{n_i \cdot t}} \end{aligned} \quad (6.35)$$

That is, we want to maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the k negative sampled non-neighbor words.

We can then use stochastic gradient descent to train to this objective, iteratively modifying the parameters (the embeddings for each target word t and each context word or noise word c in the vocabulary) to maximize the objective.

target
embedding
context
embedding

Note that the skip-gram model thus actually learns **two** separate embeddings for each word w : the **target embedding** t and the **context embedding** c . These embeddings are stored in two matrices, the **target matrix** T and the **context matrix** C . So each row i of the target matrix T is the $1 \times d$ vector embedding t_i for word i in the vocabulary V , and each column j of the context matrix C is a $d \times 1$ vector embedding c_j for word j in V . Fig. 6.12 shows an intuition of the learning task for the embeddings encoded in these two matrices.

Just as in logistic regression, then, the learning algorithm starts with randomly initialized W and C matrices, and then walks through the training corpus using gradient descent to move W and C so as to maximize the objective in Eq. 6.35. Thus the matrices W and C function as the parameters θ that logistic regression is tuning.

Once the embeddings are learned, we'll have two embeddings for each word w_i : t_i and c_i . We can choose to throw away the C matrix and just keep W , in which case each word i will be represented by the vector t_i .

Alternatively we can add the two embeddings together, using the summed embedding $t_i + c_i$ as the new d -dimensional embedding, or we can concatenate them into an embedding of dimensionality $2d$.

As with the simple count-based methods like tf-idf, the context window size L affects the performance of skip-gram embeddings, and experiments often tune the parameter L on a devset. One difference from the count-based methods is that for skip-grams, the larger the window size the more computation the algorithm requires for training (more neighboring words must be predicted).

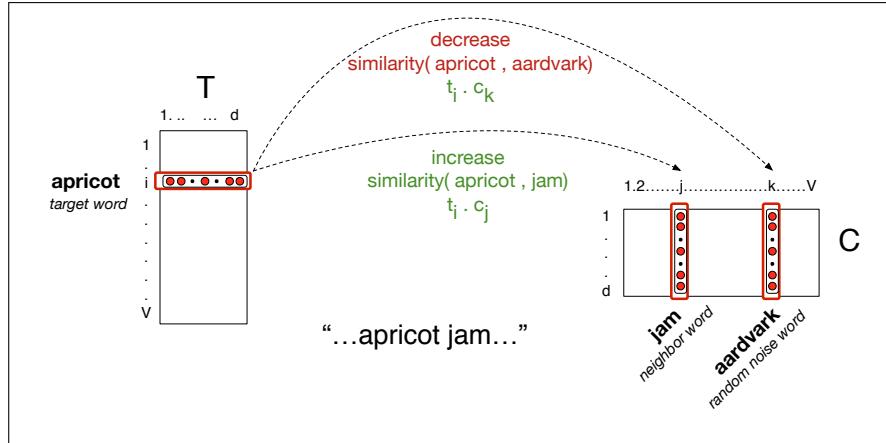


Figure 6.12 The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (have a lower dot product with) context embeddings for words that don't occur nearby (here *aardvark*).

6.9 Visualizing Embeddings

“I see well in many dimensions as long as the dimensions are around two.”

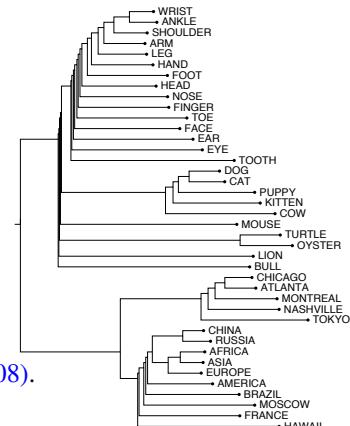
The late economist Martin Shubek

Visualizing embeddings is an important goal in helping understand, apply, and improve these models of word meaning. But how can we visualize a (for example) 100-dimensional vector?

The simplest way to visualize the meaning of a word w embedded in a space is to list the most similar words to w by sorting the vectors for all words in the vocabulary by their cosine with the vector for w . For example the 7 closest words to *frog* using the GloVe embeddings are: *frogs*, *toad*, *litoria*, *leptodactylidae*, *rana*, *lizard*, and *eleutherodactylus* (Pennington et al., 2014)

Yet another visualization method is to use a clustering algorithm to show a hierarchical representation of which words are similar to others in the embedding space. The uncaptioned example on the right uses hierarchical clustering of some embedding vectors for nouns as a visualization method (Rohde et al., 2006).

Probably the most common visualization method, however, is to project the 100 dimensions of a word down into 2 dimensions. Fig. 6.1 showed one such visualization, as does Fig. 6.13, using a projection method called t-SNE (van der Maaten and Hinton, 2008).



6.10 Semantic properties of embeddings

Vector semantic models have a number of parameters. One parameter that is relevant to both sparse tf-idf vectors and dense word2vec vectors is the size of the context

window used to collect counts. This is generally between 1 and 10 words on each side of the target word (for a total context of 3-20 words).

The choice depends on the goals of the representation. Shorter context windows tend to lead to representations that are a bit more syntactic, since the information is coming from immediately nearby words. When the vectors are computed from short context windows, the most similar words to a target word w tend to be semantically similar words with the same parts of speech. When vectors are computed from long context windows, the highest cosine words to a target word w tend to be words that are topically related but not similar.

For example Levy and Goldberg (2014a) showed that using skip-gram with a window of ± 2 , the most similar words to the word *Hogwarts* (from the *Harry Potter* series) were names of other fictional schools: *Sunnydale* (from *Buffy the Vampire Slayer*) or *Evernight* (from a vampire series). With a window of ± 5 , the most similar words to *Hogwarts* were other words topically related to the *Harry Potter* series: *Dumbledore*, *Malfoy*, and *half-blood*.

It's also often useful to distinguish two kinds of similarity or association between words (Schütze and Pedersen, 1993). Two words have **first-order co-occurrence** (sometimes called **syntagmatic association**) if they are typically nearby each other. Thus *wrote* is a first-order associate of *book* or *poem*. Two words have **second-order co-occurrence** (sometimes called **paradigmatic association**) if they have similar neighbors. Thus *wrote* is a second-order associate of words like *said* or *remarked*.

Analogy Another semantic property of embeddings is their ability to capture relational meanings. Mikolov et al. (2013b) and Levy and Goldberg (2014b) show that the *offsets* between vector embeddings can capture some analogical relations between words. For example, the result of the expression $\text{vector}(\text{'king'}) - \text{vector}(\text{'man'}) + \text{vector}(\text{'woman'})$ is a vector close to $\text{vector}(\text{'queen'})$; the left panel in Fig. 6.13 visualizes this, again projected down into 2 dimensions. Similarly, they found that the expression $\text{vector}(\text{'Paris'}) - \text{vector}(\text{'France'}) + \text{vector}(\text{'Italy'})$ results in a vector that is very close to $\text{vector}(\text{'Rome'})$.

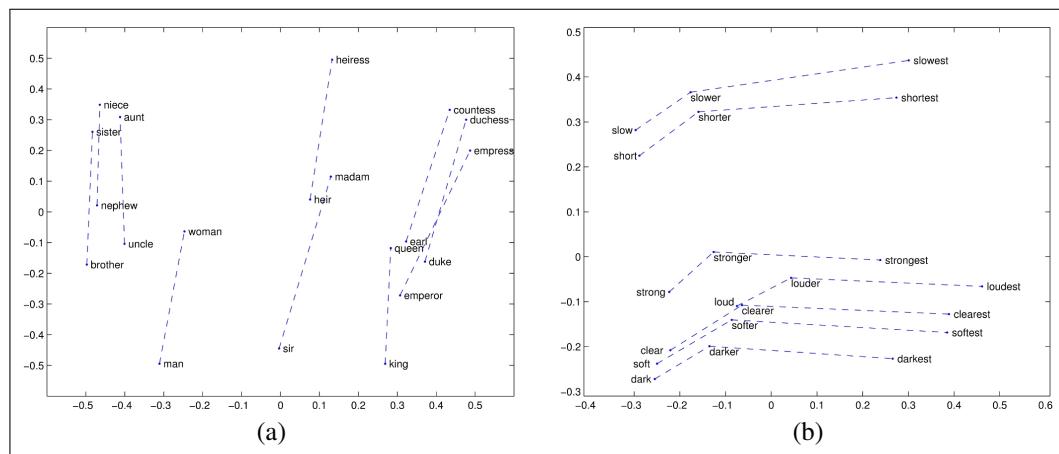


Figure 6.13 Relational properties of the vector space, shown by projecting vectors onto two dimensions. (a) '*king*' - '*man*' + '*woman*' is close to '*queen*' (b) offsets seem to capture comparative and superlative morphology (Pennington et al., 2014).

Embeddings and Historical Semantics: Embeddings can also be a useful tool for studying how meaning changes over time, by computing multiple embedding

spaces, each from texts written in a particular time period. For example Fig. 6.14 shows a visualization of changes in meaning in English words over the last two centuries, computed by building separate embedding spaces for each decade from historical corpora like Google N-grams (Lin et al., 2012) and the Corpus of Historical American English (Davies, 2012).



Figure 6.14 A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. The visualizations show the changes in the word *gay* from meanings related to “cheerful” or “frolicsome” to referring to homosexuality, the development of the modern “transmission” sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning “full of awe” to meaning “terrible or appalling” (Hamilton et al., 2016b).

6.11 Bias and Embeddings

In addition to their ability to learn word meaning from text, embeddings, alas, also reproduce the implicit biases and stereotypes that were latent in the text. Recall that embeddings model analogical relations; ‘queen’ as the closest word to ‘king’ - ‘man’ + ‘woman’ implies the analogy *man:woman::king:queen*. But embedding analogies also exhibit gender stereotypes. For example Bolukbasi et al. (2016) find that the closest occupation to ‘man’ - ‘computer programmer’ + ‘woman’ in word2vec embeddings trained on news text is ‘homemaker’, and that the embeddings similarly suggest the analogy ‘father’ is to ‘doctor’ as ‘mother’ is to ‘nurse’. Algorithms that use embeddings as part of a search for potential programmers or doctors might thus incorrectly downweight documents with women’s names.

Embeddings also encode the implicit associations that are a property of human reasoning. The Implicit Association Test (Greenwald et al., 1998) measures people’s associations between concepts (like ‘flowers’ or ‘insects’) and attributes (like ‘pleasantness’ and ‘unpleasantness’) by measuring differences in the latency with which they label words in the various categories.⁷ Using such methods, people in the United States have been shown to associate African-American names with unpleasant words (more than European-American names), male names more with

⁷ Roughly speaking, if humans associate ‘flowers’ with ‘pleasantness’ and ‘insects’ with ‘unpleasantness’, when they are instructed to push a green button for ‘flowers’ (daisy, iris, lilac) and ‘pleasant words’ (love, laughter, pleasure) and a red button for ‘insects’ (flea, spider, mosquito) and ‘unpleasant words’ (abuse, hatred, ugly) they are faster than in an incongruous condition where they push a red button for ‘flowers’ and ‘unpleasant words’ and a green button for ‘insects’ and ‘pleasant words’.

mathematics and female names with the arts, and old people’s names with unpleasant words (Greenwald et al. 1998, Nosek et al. 2002a, Nosek et al. 2002b). Caliskan et al. (2017) replicated all these findings of implicit associations using GloVe vectors and cosine similarity instead of human latencies. For example African-American names like ‘Leroy’ and ‘Shaniqua’ had a higher GloVe cosine with unpleasant words while European-American names ('Brad', 'Greg', 'Courtney') had a higher cosine with pleasant words. Any embedding-aware algorithm that made use of word sentiment could thus lead to bias against African Americans.

debiasing

Recent research focuses on ways to try to remove these kinds of biases, for example by developing a transformation of the embedding space that removes gender stereotypes but preserves definitional gender (Bolukbasi et al. 2016, Zhao et al. 2017) or changing the training procedure (Zhao et al., 2018b). However, although these sorts of **debiasing** may reduce bias in embeddings, they do not eliminate it (Gonen and Goldberg, 2019), and this remains an open problem.

Historical embeddings are also being used to measure biases in the past. Garg et al. (2018) used embeddings from historical texts to measure the association between embeddings for occupations and embeddings for names of various ethnicities or genders (for example the relative cosine similarity of women’s names versus men’s to occupation words like ‘librarian’ or ‘carpenter’) across the 20th century. They found that the cosines correlate with the empirical historical percentages of women or ethnic groups in those occupations. Historical embeddings also replicated old surveys of ethnic stereotypes; the tendency of experimental participants in 1933 to associate adjectives like ‘industrious’ or ‘superstitious’ with, e.g., Chinese ethnicity, correlates with the cosine between Chinese last names and those adjectives using embeddings trained on 1930s text. They also were able to document historical gender biases, such as the fact that embeddings for adjectives related to competence ('smart', 'wise', 'thoughtful', 'resourceful') had a higher cosine with male than female words, and showed that this bias has been slowly decreasing since 1960. We return in later chapters to this question about the role of bias in natural language processing.

6.12 Evaluating Vector Models

The most important evaluation metric for vector models is extrinsic evaluation on tasks; adding them as features into any NLP task and seeing whether this improves performance over some other model.

Nonetheless it is useful to have intrinsic evaluations. The most common metric is to test their performance on **similarity**, computing the correlation between an algorithm’s word similarity scores and word similarity ratings assigned by humans. **WordSim-353** (Finkelstein et al., 2002) is a commonly used set of ratings from 0 to 10 for 353 noun pairs; for example (*plane*, *car*) had an average score of 5.77. **SimLex-999** (Hill et al., 2015) is a more difficult dataset that quantifies similarity (*cup*, *mug*) rather than relatedness (*cup*, *coffee*), and including both concrete and abstract adjective, noun and verb pairs. The **TOEFL dataset** is a set of 80 questions, each consisting of a target word with 4 additional word choices; the task is to choose which is the correct synonym, as in the example: *Levied is closest in meaning to: imposed, believed, requested, correlated* (Landauer and Dumais, 1997). All of these datasets present words without context.

Slightly more realistic are intrinsic similarity tasks that include context. The

Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012) and the Word-in-Context (WiC) dataset (Pilehvar and Camacho-Collados, 2019) offers richer evaluation scenarios. SCWS gives human judgments on 2,003 pairs of words in their sentential context, while WiC gives target words in two sentential contexts that are either in the same or different senses; see Section 19.5.3. The *semantic textual similarity task* (Agirre et al. 2012, Agirre et al. 2015) evaluates the performance of sentence-level similarity algorithms, consisting of a set of pairs of sentences, each pair with human-labeled similarity scores.

Another task used for evaluate is an analogy task, where the system has to solve problems of the form *a is to b as c is to d*, given *a*, *b*, and *c* and having to find *d*. Thus given *Athens is to Greece as Oslo is to _____*, the system must fill in the word *Norway*. Or more syntactically-oriented examples: given *mouse*, *mice*, and *dollar* the system must return *dollars*. Large sets of such tuples have been created (Mikolov et al. 2013, Mikolov et al. 2013b).

6.13 Summary

- In vector semantics, a word is modeled as a vector—a point in high-dimensional space, also called an **embedding**.
- Vector semantic models fall into two classes: **sparse** and **dense**.
- In sparse models like **tf-idf** each dimension corresponds to a word in the vocabulary V ; cells in sparse models are functions of **co-occurrence counts**. The **term-document** matrix has rows for each word (**term**) in the vocabulary and a column for each document. The **word-context** matrix has a row for each (target) word in the vocabulary and a column for each context term in the vocabulary.
- The most widely used sparse weighting is **tf-idf**, which weights each cell by its **term frequency** and **inverse document frequency**. **PPMI** (pointwise positive mutual information) is an alternative weighting scheme to tf-idf.
- Dense vector models have dimensionality 50–1000 and the dimensions are harder to interpret. **Word2vec** algorithms like **skip-gram** are a popular and efficient way to compute dense embeddings. Skip-gram trains a logistic regression classifier to compute the probability that two words are ‘likely to occur nearby in text’. This probability is computed from the dot product between the embeddings for the two words.
- Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.
- Other important embedding algorithms include **GloVe**, a method based on ratios of word co-occurrence probabilities, and **fasttext**, an open-source library for computing word embeddings by summing embeddings of the bag of character n-grams that make up a word.
- Whether using sparse or dense vectors, word and document similarities are computed by some function of the **dot product** between vectors. The cosine of two vectors—a normalized dot product—is the most popular such metric.

Bibliographical and Historical Notes

The idea of vector semantics arose out of research in the 1950s in three distinct fields: linguistics, psychology, and computer science, each of which contributed a fundamental aspect of the model.

The idea that meaning is related to the distribution of words in context was widespread in linguistic theory of the 1950s, among distributionalists like Zellig Harris, Martin Joos, and J. R. Firth, and semioticians like Thomas Sebeok. As [Joos \(1950\)](#) put it,

the linguist’s “meaning” of a morpheme... is by definition the set of conditional probabilities of its occurrence in context with all other morphemes.

The idea that the meaning of a word might be modeled as a point in a multi-dimensional semantic space came from psychologists like Charles E. Osgood, who had been studying how people responded to the meaning of words by assigning values along scales like *happy/sad* or *hard/soft*. [Osgood et al. \(1957\)](#) proposed that the meaning of a word in general could be modeled as a point in a multidimensional Euclidean space, and that the similarity of meaning between two words could be modeled as the distance between these points in the space.

mechanical indexing

A final intellectual source in the 1950s and early 1960s was the field then called **mechanical indexing**, now known as **information retrieval**. In what became known as the **vector space model** for information retrieval ([Salton 1971](#), [Sparck Jones 1986](#)), researchers demonstrated new ways to define the meaning of words in terms of vectors ([Switzer, 1965](#)), and refined methods for word similarity based on measures of statistical association between words like mutual information ([Giuliano, 1965](#)) and idf ([Sparck Jones, 1972](#)), and showed that the meaning of documents could be represented in the same vector spaces used for words.

semantic feature

More distantly related is the idea of defining words by a vector of discrete features, which has a venerable history in our field, with roots at least as far back as Descartes and Leibniz ([Wierzbicka 1992](#), [Wierzbicka 1996](#)). By the middle of the 20th century, beginning with the work of Hjelmslev ([Hjelmslev, 1969](#)) and fleshed out in early models of generative grammar ([Katz and Fodor, 1963](#)), the idea arose of representing meaning with **semantic features**, symbols that represent some sort of primitive meaning. For example words like *hen*, *rooster*, or *chick*, have something in common (they all describe chickens) and something different (their age and sex), representable as:

<i>hen</i>	+female, +chicken, +adult
<i>rooster</i>	-female, +chicken, +adult
<i>chick</i>	+chicken, -adult

The dimensions used by vector models of meaning to define words, however, are only abstractly related to this idea of a small fixed number of hand-built dimensions. Nonetheless, there has been some attempt to show that certain dimensions of embedding models do contribute some specific compositional aspect of meaning like these early semantic features.

SVD

The first use of dense vectors to model word meaning was the **latent semantic indexing** (LSI) model ([Deerwester et al., 1988](#)) recast as **LSA** (**latent semantic analysis**) ([Deerwester et al., 1990](#)). In **LSA singular value decomposition**—**SVD**—is applied to a term-document matrix (each cell weighted by log frequency and normalized by entropy), and then the first 300 dimensions are used as the LSA

embedding. Singular Value Decomposition (SVD) is a method for finding the most important dimensions of a data set, those dimensions along which the data varies the most. LSA was then quickly widely applied: as a cognitive model [Landauer and Dumais \(1997\)](#), and tasks like spell checking ([Jones and Martin, 1997](#)), language modeling ([Bellegarda 1997, Coccaro and Jurafsky 1998, Bellegarda 2000](#)) morphology induction ([Schone and Jurafsky 2000, Schone and Jurafsky 2001](#)), and essay grading ([Rehder et al., 1998](#)). Related models were simultaneously developed and applied to word sense disambiguation by [Schütze \(1992b\)](#). LSA also led to the earliest use of embeddings to represent words in a probabilistic classifier, in the logistic regression document router of [Schütze et al. \(1995\)](#). The idea of SVD on the term-term matrix (rather than the term-document matrix) as a model of meaning for NLP was proposed soon after LSA by [Schütze \(1992b\)](#). Schütze applied the low-rank (97-dimensional) embeddings produced by SVD to the task of word sense disambiguation, analyzed the resulting semantic space, and also suggested possible techniques like dropping high-order dimensions. See [Schütze \(1997a\)](#).

A number of alternative matrix models followed on from the early SVD work, including Probabilistic Latent Semantic Indexing (PLSI) ([Hofmann, 1999](#)), Latent Dirichlet Allocation (LDA) ([Blei et al., 2003](#)), and Non-negative Matrix Factorization (NMF) ([Lee and Seung, 1999](#)).

By the next decade, [Bengio et al. \(2003\)](#) and [Bengio et al. \(2006\)](#) showed that neural language models could also be used to develop embeddings as part of the task of word prediction. [Collobert and Weston \(2007, Collobert and Weston \(2008\)](#), and [Collobert et al. \(2011\)](#) then demonstrated that embeddings could play a role for representing word meanings for a number of NLP tasks. [Turian et al. \(2010\)](#) compared the value of different kinds of embeddings for different NLP tasks. [Mikolov et al. \(2011\)](#) showed that recurrent neural nets could be used as language models. The idea of simplifying the hidden layer of these neural net language models to create the skip-gram (and also CBOW) algorithms was proposed by [Mikolov et al. \(2013\)](#). The negative sampling training algorithm was proposed in [Mikolov et al. \(2013a\)](#).

Studies of embeddings include results showing an elegant mathematical relationship between sparse and dense embeddings ([Levy and Goldberg, 2014c](#)), as well as numerous surveys of embeddings and their parameterizations. ([Bullinaria and Levy 2007, Bullinaria and Levy 2012, Lapesa and Evert 2014, Kiela and Clark 2014, Levy et al. 2015](#)).

The most widely-used embedding model besides word2vec is GloVe ([Pennington et al., 2014](#)). The name stands for Global Vectors, because the model is based on capturing global corpus statistics. GloVe is based on ratios of probabilities from the word-word co-occurrence matrix, combining the intuitions of count-based models like PPMI while also capturing the linear structures used by methods like word2vec.

fasttext

An extension of word2vec, **fasttext** ([Bojanowski et al., 2017](#)), deals with unknown words and sparsity in languages with rich morphology, by using subword models. Each word in fasttext is represented as itself plus a bag of constituent n-grams, with special boundary symbols < and > added to each word. For example, with $n = 3$ the word *where* would be represented by the character n-grams:

<wh, whe, her, ere, re>

plus the sequence

<where>

Then a skipgram embedding is learned for each constituent n-gram, and the word *where* is represented by the sum of all of the embeddings of its constituent n-grams.

A fasttext open-source library, including pretrained embeddings for 157 languages, is available at <https://fasttext.cc>.

There are many other embedding algorithms, using methods like non-negative matrix factorization (Fyshe et al., 2015), or by converting sparse PPMI embeddings to dense vectors by using SVD (Levy and Goldberg, 2014c).

In Chapter 10 we introduce **contextual embeddings** like ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019) in which the representation for a word is contextual, a function of the entire input sentence.

See Manning et al. (2008) for a deeper understanding of the role of vectors in information retrieval, including how to compare queries with documents, more details on tf-idf, and issues of scaling to very large datasets.

Cruse (2004) is a useful introductory linguistic text on lexical semantics.

Exercises

Neural Networks and Neural Language Models

“[M]achines of this character can behave in a very complicated manner when the number of units is large.”

Alan Turing (1948) “Intelligent Machines”, page 6

Neural networks are a fundamental computational tool for language processing, and a very old one. They are called neural because their origins lie in the **McCulloch-Pitts neuron** (McCulloch and Pitts, 1943), a simplified model of the human neuron as a kind of computing element that could be described in terms of propositional logic. But the modern use in language processing no longer draws on these early biological inspirations.

Instead, a modern neural network is a network of small computing units, each of which takes a vector of input values and produces a single output value. In this chapter we introduce the neural net applied to classification. The architecture we introduce is called a **feedforward network** because the computation proceeds iteratively from one layer of units to the next. The use of modern neural nets is often called **deep learning**, because modern networks are often **deep** (have many layers).

feedforward

deep learning

Neural networks share much of the same mathematics as logistic regression. But neural networks are a more powerful classifier than logistic regression, and indeed a minimal neural network (technically one with a single ‘hidden layer’) can be shown to learn any function.

Neural net classifiers are different from logistic regression in another way. With logistic regression, we applied the regression classifier to many different tasks by developing many rich kinds of feature templates based on domain knowledge. When working with neural networks, it is more common to avoid most uses of rich hand-derived features, instead building neural networks that take raw words as inputs and learn to induce features as part of the process of learning to classify. We saw examples of this kind of representation learning for embeddings in Chapter 6. Nets that are very deep are particularly good at representation learning. For that reason deep neural nets are the right tool for large scale problems that offer sufficient data to learn features automatically.

In this chapter we’ll introduce feedforward networks as classifiers, and also apply them to the simple task of language modeling: assigning probabilities to word sequences and predicting upcoming words. In subsequent chapters we’ll introduce many other aspects of neural models, such as **recurrent neural networks** (Chapter 9), **encoder-decoder** models, **attention** and the **Transformer** (Chapter 10).

7.1 Units

The building block of a neural network is a single computational unit. A unit takes a set of real valued numbers as input, performs some computation on them, and produces an output.

bias term At its heart, a neural unit is taking a weighted sum of its inputs, with one additional term in the sum called a **bias term**. Given a set of inputs $x_1 \dots x_n$, a unit has a set of corresponding weights $w_1 \dots w_n$ and a bias b , so the weighted sum z can be represented as:

$$z = b + \sum_i w_i x_i \quad (7.1)$$

vector Often it's more convenient to express this weighted sum using vector notation; recall from linear algebra that a **vector** is, at heart, just a list or array of numbers. Thus we'll talk about z in terms of a weight vector w , a scalar bias b , and an input vector x , and we'll replace the sum with the convenient **dot product**:

$$z = w \cdot x + b \quad (7.2)$$

As defined in Eq. 7.2, z is just a real valued number.

activation Finally, instead of using z , a linear function of x , as the output, neural units apply a non-linear function f to z . We will refer to the output of this function as the **activation** value for the unit, a . Since we are just modeling a single unit, the activation for the node is in fact the final output of the network, which we'll generally call y . So the value y is defined as:

$$y = a = f(z)$$

We'll discuss three popular non-linear functions $f()$ below (the sigmoid, the tanh, and the rectified linear ReLU) but it's pedagogically convenient to start with the **sigmoid** function since we saw it in Chapter 5:

$$y = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (7.3)$$

The sigmoid (shown in Fig. 7.1) has a number of advantages; it maps the output into the range $[0, 1]$, which is useful in squashing outliers toward 0 or 1. And it's differentiable, which as we saw in Section 5.8 will be handy for learning.

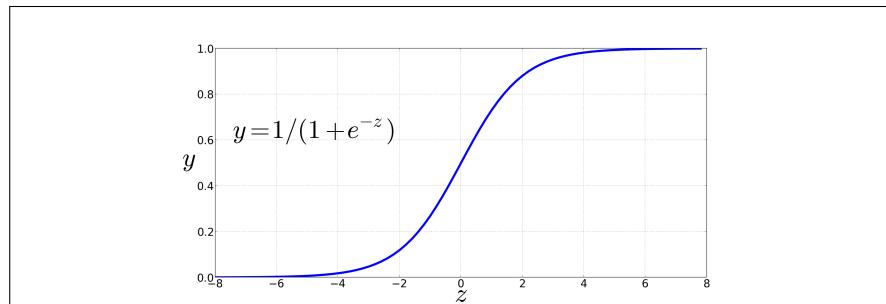


Figure 7.1 The sigmoid function takes a real value and maps it to the range $[0, 1]$. It is nearly linear around 0 but outlier values get squashed toward 0 or 1.

Substituting Eq. 7.2 into Eq. 7.3 gives us the output of a neural unit:

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))} \quad (7.4)$$

Fig. 7.2 shows a final schematic of a basic neural unit. In this example the unit takes 3 input values x_1, x_2 , and x_3 , and computes a weighted sum, multiplying each value by a weight (w_1, w_2 , and w_3 , respectively), adds them to a bias term b , and then passes the resulting sum through a sigmoid function to result in a number between 0 and 1.

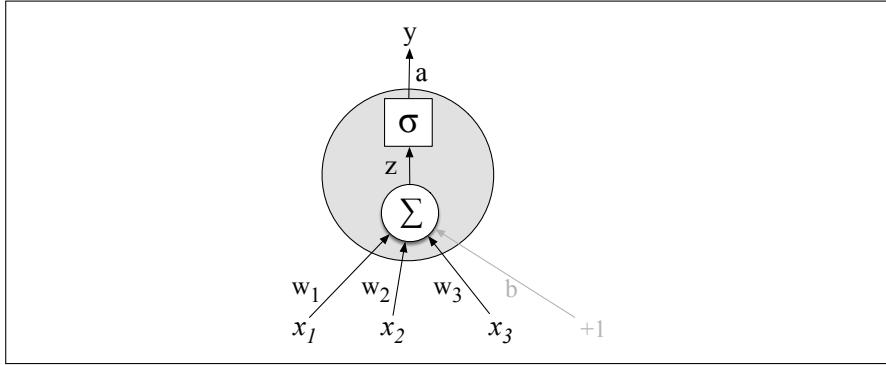


Figure 7.2 A neural unit, taking 3 inputs x_1, x_2 , and x_3 (and a bias b that we represent as a weight for an input clamped at +1) and producing an output y . We include some convenient intermediate variables: the output of the summation, z , and the output of the sigmoid, a . In this case the output of the unit y is the same as a , but in deeper networks we'll reserve y to mean the final output of the entire network, leaving a as the activation of an individual node.

Let's walk through an example just to get an intuition. Let's suppose we have a unit with the following weight vector and bias:

$$\begin{aligned} w &= [0.2, 0.3, 0.9] \\ b &= 0.5 \end{aligned}$$

What would this unit do with the following input vector:

$$x = [0.5, 0.6, 0.1]$$

The resulting output y would be:

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = e^{-0.87} = .70$$

tanh In practice, the sigmoid is not commonly used as an activation function. A function that is very similar but almost always better is the **tanh** function shown in Fig. 7.3a; tanh is a variant of the sigmoid that ranges from -1 to +1:

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (7.5)$$

ReLU The simplest activation function, and perhaps the most commonly used, is the rectified linear unit, also called the **ReLU**, shown in Fig. 7.3b. It's just the same as x when x is positive, and 0 otherwise:

$$y = \max(x, 0) \quad (7.6)$$

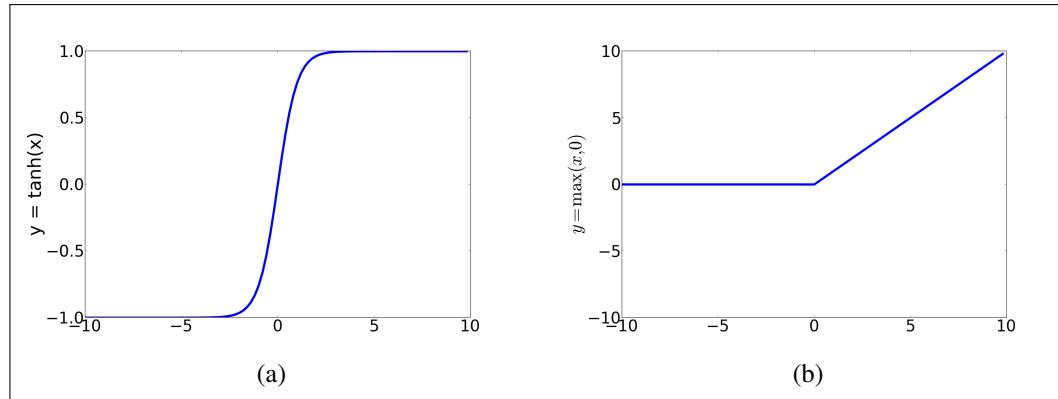


Figure 7.3 The tanh and ReLU activation functions.

saturated

These activation functions have different properties that make them useful for different language applications or network architectures. For example the rectifier function has nice properties that result from it being very close to linear. In the sigmoid or tanh functions, very high values of z result in values of y that are **saturated**, i.e., extremely close to 1, which causes problems for learning. Rectifiers don't have this problem, since the output of values close to 1 also approaches 1 in a nice gentle linear way. By contrast, the tanh function has the nice properties of being smoothly differentiable and mapping outlier values toward the mean.

7.2 The XOR problem

Early in the history of neural networks it was realized that the power of neural networks, as with the real neurons that inspired them, comes from combining these units into larger networks.

One of the most clever demonstrations of the need for multi-layer networks was the proof by [Minsky and Papert \(1969\)](#) that a single neural unit cannot compute some very simple functions of its input. Consider the task of computing elementary logical functions of two inputs, like AND, OR, and XOR. As a reminder, here are the truth tables for those functions:

		AND		OR		XOR		
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

perceptron

This example was first shown for the **perceptron**, which is a very simple neural unit that has a binary output and does **not** have a non-linear activation function. The output y of a perceptron is 0 or 1, and is computed as follows (using the same weight w , input x , and bias b as in Eq. 7.2):

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases} \quad (7.7)$$

It's very easy to build a perceptron that can compute the logical AND and OR functions of its binary inputs; Fig. 7.4 shows the necessary weights.

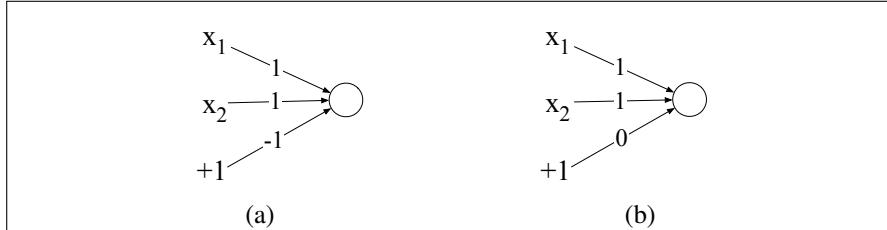


Figure 7.4 The weights w and bias b for perceptrons for computing logical functions. The inputs are shown as x_1 and x_2 and the bias as a special node with value +1 which is multiplied with the bias weight b . (a) logical AND, showing weights $w_1 = 1$ and $w_2 = 1$ and bias weight $b = -1$. (b) logical OR, showing weights $w_1 = 1$ and $w_2 = 1$ and bias weight $b = 0$. These weights/biases are just one from an infinite number of possible sets of weights and biases that would implement the functions.

It turns out, however, that it's not possible to build a perceptron to compute logical XOR! (It's worth spending a moment to give it a try!)

The intuition behind this important result relies on understanding that a perceptron is a linear classifier. For a two-dimensional input x_1 and x_2 , the perception equation, $w_1x_1 + w_2x_2 + b = 0$ is the equation of a line. (We can see this by putting it in the standard linear format: $x_2 = -(w_1/w_2)x_1 - b$.) This line acts as a **decision boundary** in two-dimensional space in which the output 0 is assigned to all inputs lying on one side of the line, and the output 1 to all input points lying on the other side of the line. If we had more than 2 inputs, the decision boundary becomes a hyperplane instead of a line, but the idea is the same, separating the space into two categories.

Fig. 7.5 shows the possible logical inputs (00, 01, 10, and 11) and the line drawn by one possible set of parameters for an AND and an OR classifier. Notice that there is simply no way to draw a line that separates the positive cases of XOR (01 and 10) from the negative cases (00 and 11). We say that XOR is not a **linearly separable** function. Of course we could draw a boundary with a curve, or some other function, but not a single line.

decision boundary

linearly separable

7.2.1 The solution: neural networks

While the XOR function cannot be calculated by a single perceptron, it can be calculated by a layered network of units. Let's see an example of how to do this from Goodfellow et al. (2016) that computes XOR using two layers of ReLU-based units. Fig. 7.6 shows a figure with the input being processed by two layers of neural units. The middle layer (called h) has two units, and the output layer (called y) has one unit. A set of weights and biases are shown for each ReLU that correctly computes the XOR function.

Let's walk through what happens with the input $x = [0\ 0]$. If we multiply each input value by the appropriate weight, sum, and then add the bias b , we get the vector $[0\ -1]$, and we then apply the rectified linear transformation to give the output of the h layer as $[0\ 0]$. Now we once again multiply by the weights, sum, and add the bias (0 in this case) resulting in the value 0. The reader should work through the computation of the remaining 3 possible input pairs to see that the resulting y values are 1 for the inputs $[0\ 1]$ and $[1\ 0]$ and 0 for $[0\ 0]$ and $[1\ 1]$.



Figure 7.5 The functions AND, OR, and XOR, represented with input x_1 on the x-axis and input x_2 on the y axis. Filled circles represent perceptron outputs of 1, and white circles perceptron outputs of 0. There is no way to draw a line that correctly separates the two categories for XOR. Figure styled after [Russell and Norvig \(2002\)](#).

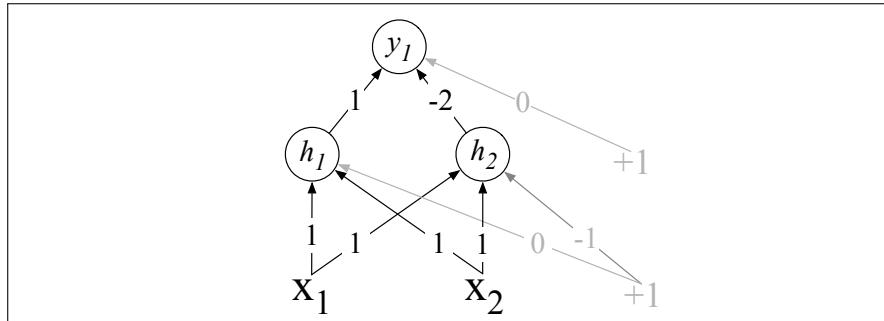


Figure 7.6 XOR solution after [Goodfellow et al. \(2016\)](#). There are three ReLU units, in two layers; we've called them h_1 , h_2 (h for “hidden layer”) and y_1 . As before, the numbers on the arrows represent the weights w for each unit, and we represent the bias b as a weight on a unit clamped to +1, with the bias weights/units in gray.

It's also instructive to look at the intermediate results, the outputs of the two hidden nodes h_0 and h_1 . We showed in the previous paragraph that the h vector for the inputs $x = [0 0]$ was $[0 0]$. Fig. 7.7b shows the values of the h layer for all 4 inputs. Notice that hidden representations of the two input points $x = [0 1]$ and $x = [1 0]$ (the two cases with XOR output = 1) are merged to the single point $h = [1 0]$. The merger makes it easy to linearly separate the positive and negative cases of XOR. In other words, we can view the hidden layer of the network as forming a representation for the input.

In this example we just stipulated the weights in Fig. 7.6. But for real examples the weights for neural networks are learned automatically using the error backpropagation algorithm to be introduced in Section 7.4. That means the hidden layers will learn to form useful representations. This intuition, that neural networks can automatically learn useful representations of the input, is one of their key advantages, and one that we will return to again and again in later chapters.

Note that the solution to the XOR problem requires a network of units with non-linear activation functions. A network made up of simple linear (perceptron) units cannot solve the XOR problem. This is because a network formed by many layers of purely linear units can always be reduced (shown to be computationally identical

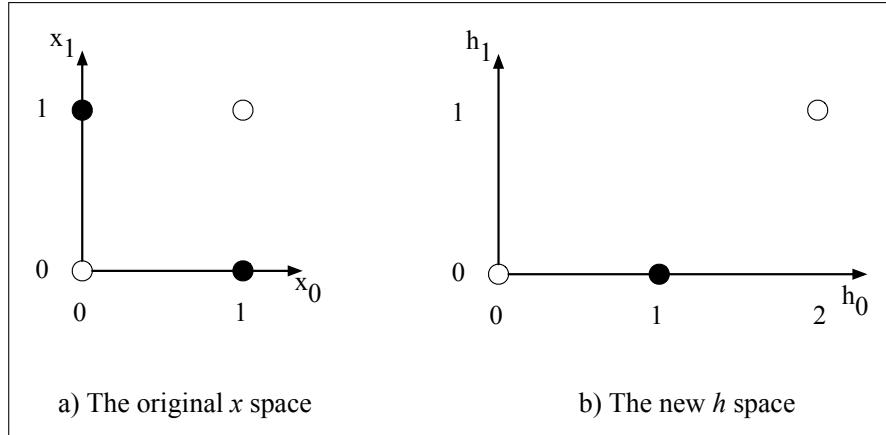


Figure 7.7 The hidden layer forming a new representation of the input. Here is the representation of the hidden layer, h , compared to the original input representation x . Notice that the input point $[0 \ 1]$ has been collapsed with the input point $[1 \ 0]$, making it possible to linearly separate the positive and negative cases of XOR. After Goodfellow et al. (2016).

to a single layer of linear units with appropriate weights, and we've already shown (visually, in Fig. 7.5) that a single unit cannot solve the XOR problem.

7.3 Feed-Forward Neural Networks

feedforward network

Let's now walk through a slightly more formal presentation of the simplest kind of neural network, the **feedforward network**. A feedforward network is a multilayer network in which the units are connected with no cycles; the outputs from units in each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers. (In Chapter 9 we'll introduce networks with cycles, called **recurrent neural networks**.)

multi-layer perceptrons
MLP

For historical reasons multilayer networks, especially feedforward networks, are sometimes called **multi-layer perceptrons** (or MLPs); this is a technical misnomer, since the units in modern multilayer networks aren't perceptrons (perceptrons are purely linear, but modern networks are made up of units with non-linearities like sigmoids), but at some point the name stuck.

hidden layer

Simple feedforward networks have three kinds of nodes: input units, hidden units, and output units. Fig. 7.8 shows a picture.

fully-connected

The input units are simply scalar values just as we saw in Fig. 7.2.

The core of the neural network is the **hidden layer** formed of **hidden units**, each of which is a neural unit as described in Section 7.1, taking a weighted sum of its inputs and then applying a non-linearity. In the standard architecture, each layer is **fully-connected**, meaning that each unit in each layer takes as input the outputs from all the units in the previous layer, and there is a link between every pair of units from two adjacent layers. Thus each hidden unit sums over all the input units.

Recall that a single hidden unit has parameters w (the weight vector) and b (the bias scalar). We represent the parameters for the entire hidden layer by combining the weight vector \mathbf{w}_i and bias b_i for each unit i into a single weight matrix W and a single bias vector \mathbf{b} for the whole layer (see Fig. 7.8). Each element W_{ij} of the weight matrix W represents the weight of the connection from the i th input unit x_i to

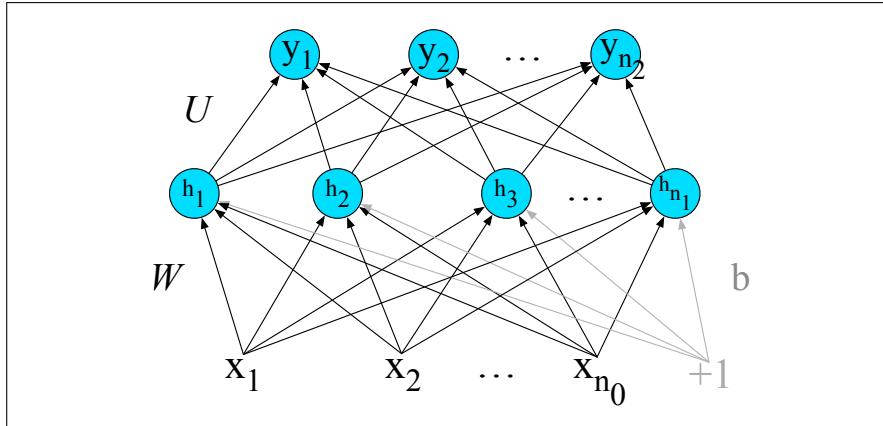


Figure 7.8 A simple 2-layer feedforward network, with one hidden layer, one output layer, and one input layer (the input layer is usually not counted when enumerating layers).

the j th hidden unit h_j .

The advantage of using a single matrix W for the weights of the entire layer is that now the hidden layer computation for a feedforward network can be done very efficiently with simple matrix operations. In fact, the computation only has three steps: multiplying the weight matrix by the input vector x , adding the bias vector b , and applying the activation function g (such as the sigmoid, tanh, or ReLU activation function defined above).

The output of the hidden layer, the vector h , is thus the following, using the sigmoid function σ :

$$h = \sigma(Wx + b) \quad (7.8)$$

Notice that we're applying the σ function here to a vector, while in Eq. 7.3 it was applied to a scalar. We're thus allowing $\sigma(\cdot)$, and indeed any activation function $g(\cdot)$, to apply to a vector element-wise, so $g[z_1, z_2, z_3] = [g(z_1), g(z_2), g(z_3)]$.

Let's introduce some constants to represent the dimensionalities of these vectors and matrices. We'll refer to the input layer as layer 0 of the network, and have n_0 represent the number of inputs, so x is a vector of real numbers of dimension n_0 , or more formally $x \in \mathbb{R}^{n_0}$. Let's call the hidden layer layer 1 and the output layer layer 2. The hidden layer has dimensionality n_1 , so $h \in \mathbb{R}^{n_1}$ and also $b \in \mathbb{R}^{n_1}$ (since each hidden unit can take a different bias value). And the weight matrix W has dimensionality $W \in \mathbb{R}^{n_1 \times n_0}$.

Take a moment to convince yourself that the matrix multiplication in Eq. 7.8 will compute the value of each h_j as $\sigma(\sum_{i=1}^{n_0} w_{ij}x_i + b_j)$.

As we saw in Section 7.2, the resulting value h (for *hidden* but also for *hypothesis*) forms a *representation* of the input. The role of the output layer is to take this new representation h and compute a final output. This output could be a real-valued number, but in many cases the goal of the network is to make some sort of classification decision, and so we will focus on the case of classification.

If we are doing a binary task like sentiment classification, we might have a single output node, and its value y is the probability of positive versus negative sentiment. If we are doing multinomial classification, such as assigning a part-of-speech tag, we might have one output node for each potential part-of-speech, whose output value is the probability of that part-of-speech, and the values of all the output nodes must sum to one. The output layer thus gives a probability distribution across the output

nodes.

Let's see how this happens. Like the hidden layer, the output layer has a weight matrix (let's call it U), but some models don't include a bias vector b in the output layer, so we'll simplify by eliminating the bias vector in this example. The weight matrix is multiplied by its input vector (h) to produce the intermediate output z .

$$z = Uh$$

There are n_2 output nodes, so $z \in \mathbb{R}^{n_2}$, weight matrix U has dimensionality $U \in \mathbb{R}^{n_2 \times n_1}$, and element U_{ij} is the weight from unit j in the hidden layer to unit i in the output layer.

normalizing

softmax

However, z can't be the output of the classifier, since it's a vector of real-valued numbers, while what we need for classification is a vector of probabilities. There is a convenient function for **normalizing** a vector of real values, by which we mean converting it to a vector that encodes a probability distribution (all the numbers lie between 0 and 1 and sum to 1): the **softmax** function that we saw on page 89 of Chapter 5. For a vector z of dimensionality d , the softmax is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^d e^{z_j}} \quad 1 \leq i \leq d \quad (7.9)$$

Thus for example given a vector $z=[0.6 \ 1.1 \ -1.5 \ 1.2 \ 3.2 \ -1.1]$, $\text{softmax}(z)$ is $[0.055 \ 0.090 \ 0.0067 \ 0.10 \ 0.74 \ 0.010]$.

You may recall that softmax was exactly what is used to create a probability distribution from a vector of real-valued numbers (computed from summing weights times features) in logistic regression in Chapter 5.

That means we can think of a neural network classifier with one hidden layer as building a vector h which is a hidden layer representation of the input, and then running standard logistic regression on the features that the network develops in h . By contrast, in Chapter 5 the features were mainly designed by hand via feature templates. So a neural network is like logistic regression, but (a) with many layers, since a deep neural network is like layer after layer of logistic regression classifiers, and (b) rather than forming the features by feature templates, the prior layers of the network induce the feature representations themselves.

Here are the final equations for a feedforward network with a single hidden layer, which takes an input vector x , outputs a probability distribution y , and is parameterized by weight matrices W and U and a bias vector b :

$$\begin{aligned} h &= \sigma(Wx + b) \\ z &= Uh \\ y &= \text{softmax}(z) \end{aligned} \quad (7.10)$$

We'll call this network a 2-layer network (we traditionally don't count the input layer when numbering layers, but do count the output layer). So by this terminology logistic regression is a 1-layer network.

Let's now set up some notation to make it easier to talk about deeper networks of depth more than 2. We'll use superscripts in square brackets to mean layer numbers, starting at 0 for the input layer. So $W^{[1]}$ will mean the weight matrix for the (first) hidden layer, and $b^{[1]}$ will mean the bias vector for the (first) hidden layer. n_j will mean the number of units at layer j . We'll use $g(\cdot)$ to stand for the activation function, which will tend to be ReLU or tanh for intermediate layers and softmax for output layers. We'll use $a^{[i]}$ to mean the output from layer i , and $z^{[i]}$ to mean the

combination of weights and biases $W^{[i]}a^{[i-1]} + b^{[i]}$. The 0th layer is for inputs, so the inputs x we'll refer to more generally as $a^{[0]}$.

Thus we can re-represent our 2-layer net from Eq. 7.10 as follows:

$$\begin{aligned} z^{[1]} &= W^{[1]}a^{[0]} + b^{[1]} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= g^{[2]}(z^{[2]}) \\ \hat{y} &= a^{[2]} \end{aligned} \tag{7.11}$$

Note that with this notation, the equations for the computation done at each layer are the same. The algorithm for computing the forward step in an n-layer feedforward network, given the input vector $a^{[0]}$ is thus simply:

```
for i in 1..n
     $z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}$ 
     $a^{[i]} = g^{[i]}(z^{[i]})$ 
 $\hat{y} = a^{[n]}$ 
```

The activation functions $g(\cdot)$ are generally different at the final layer. Thus $g^{[2]}$ might be softmax for multinomial classification or sigmoid for binary classification, while ReLU or tanh might be the activation function $g(\cdot)$ at the internal layers.

7.4 Training Neural Nets

A feedforward neural net is an instance of supervised machine learning in which we know the correct output y for each observation x . What the system produces, via Eq. 7.11, is \hat{y} , the system's estimate of the true y . The goal of the training procedure is to learn parameters $W^{[i]}$ and $b^{[i]}$ for each layer i that make \hat{y} for each training observation as close as possible to the true y .

In general, we do all this by drawing on the methods we introduced in Chapter 5 for logistic regression, so the reader should be comfortable with that chapter before proceeding.

First, we'll need a **loss function** that models the distance between the system output and the gold output, and it's common to use the loss function used for logistic regression, the **cross-entropy loss**.

Second, to find the parameters that minimize this loss function, we'll use the **gradient descent** optimization algorithm introduced in Chapter 5.

Third, gradient descent requires knowing the **gradient** of the loss function, the vector that contains the partial derivative of the loss function with respect to each of the parameters. Here is one part where learning for neural networks is more complex than for logistic regression. In logistic regression, for each observation we could directly compute the derivative of the loss function with respect to an individual w or b . But for neural networks, with millions of parameters in many layers, it's much harder to see how to compute the partial derivative of some weight in layer 1 when the loss is attached to some much later layer. How do we partial out the loss over all those intermediate layers?

The answer is the algorithm called **error backpropagation** or **reverse differentiation**.

7.4.1 Loss function

cross-entropy loss

The **cross-entropy loss** that is used in neural networks is the same one we saw for logistic regression.

In fact, if the neural network is being used as a binary classifier, with the sigmoid at the final layer, the loss function is exactly the same as we saw with logistic regression in Eq. 5.10:

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y) \log(1-\hat{y})] \quad (7.12)$$

What about if the neural network is being used as a multinomial classifier? Let y be a vector over the C classes representing the true output probability distribution. The cross-entropy loss here is

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^C y_i \log \hat{y}_i \quad (7.13)$$

negative log likelihood loss

We can simplify this equation further. Assume this is a **hard classification** task, meaning that only one class is the correct one, and that there is one output unit in y for each class. If the true class is i , then y is a vector where $y_i = 1$ and $y_j = 0 \ \forall j \neq i$. A vector like this, with one value=1 and the rest 0, is called a **one-hot vector**. Now let \hat{y} be the vector output from the network. The sum in Eq. 7.13 will be 0 except for the true class. Hence the cross-entropy loss is simply the log probability of the correct class, and we therefore also call this the **negative log likelihood loss**:

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i \quad (7.14)$$

Plugging in the softmax formula from Eq. 7.9, and with K the number of classes:

$$L_{CE}(\hat{y}, y) = -\log \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (7.15)$$

7.4.2 Computing the Gradient

How do we compute the gradient of this loss function? Computing the gradient requires the partial derivative of the loss function with respect to each parameter. For a network with one weight layer and sigmoid output (which is what logistic regression is), we could simply use the derivative of the loss that we used for logistic regression in Eq. 7.16 (and derived in Section 5.8):

$$\begin{aligned} \frac{\partial L_{CE}(w, b)}{\partial w_j} &= (\hat{y} - y) x_j \\ &= (\sigma(w \cdot x + b) - y) x_j \end{aligned} \quad (7.16)$$

Or for a network with one hidden layer and softmax output, we could use the derivative of the softmax loss from Eq. 5.35:

$$\begin{aligned} \frac{\partial L_{CE}}{\partial w_k} &= (1\{y=k\} - p(y=k|x))x_k \\ &= \left(1\{y=k\} - \frac{e^{w_k \cdot x + b_k}}{\sum_{j=1}^K e^{w_j \cdot x + b_j}} \right) x_k \end{aligned} \quad (7.17)$$

But these derivatives only give correct updates for one weight layer: the last one! For deep networks, computing the gradients for each weight is much more complex, since we are computing the derivative with respect to weight parameters that appear all the way back in the very early layers of the network, even though the loss is computed only at the very end of the network.

error back-propagation

The solution to computing this gradient is an algorithm called **error backpropagation** or **backprop** (Rumelhart et al., 1986). While backprop was invented specifically for neural networks, it turns out to be the same as a more general procedure called **backward differentiation**, which depends on the notion of **computation graphs**. Let's see how that works in the next subsection.

7.4.3 Computation Graphs

A computation graph is a representation of the process of computing a mathematical expression, in which the computation is broken down into separate operations, each of which is modeled as a node in a graph.

Consider computing the function $L(a, b, c) = c(a + 2b)$. If we make each of the component addition and multiplication operations explicit, and add names (d and e) for the intermediate outputs, the resulting series of computations is:

$$\begin{aligned} d &= 2 * b \\ e &= a + d \\ L &= c * e \end{aligned}$$

We can now represent this as a graph, with nodes for each operation, and directed edges showing the outputs from each operation as the inputs to the next, as in Fig. 7.9. The simplest use of computation graphs is to compute the value of the function with some given inputs. In the figure, we've assumed the inputs $a = 3$, $b = 1$, $c = -2$, and we've shown the result of the **forward pass** to compute the result $L(3, 1, -2) = 10$. In the forward pass of a computation graph, we apply each operation left to right, passing the outputs of each computation as the input to the next node.

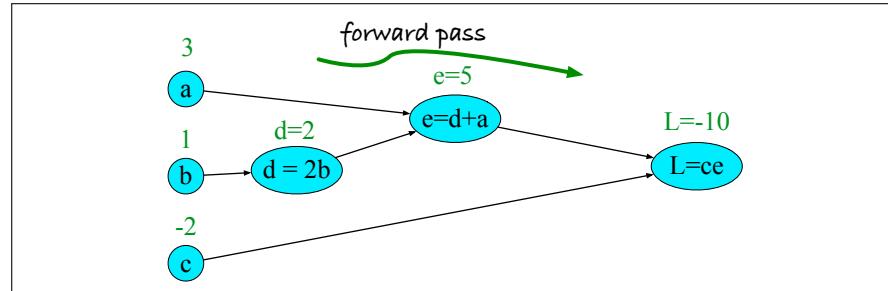


Figure 7.9 Computation graph for the function $L(a, b, c) = c(a + 2b)$, with values for input nodes $a = 3$, $b = 1$, $c = -2$, showing the forward pass computation of L .

7.4.4 Backward differentiation on computation graphs

The importance of the computation graph comes from the **backward pass**, which is used to compute the derivatives that we'll need for the weight update. In this example our goal is to compute the derivative of the output function L with respect

to each of the input variables, i.e., $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$. The derivative $\frac{\partial L}{\partial a}$, tells us how much a small change in a affects L .

chain rule

Backwards differentiation makes use of the **chain rule** in calculus. Suppose we are computing the derivative of a composite function $f(x) = u(v(x))$. The derivative of $f(x)$ is the derivative of $u(x)$ with respect to $v(x)$ times the derivative of $v(x)$ with respect to x :

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx} \quad (7.18)$$

The chain rule extends to more than two functions. If computing the derivative of a composite function $f(x) = u(v(w(x)))$, the derivative of $f(x)$ is:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dw} \cdot \frac{dw}{dx} \quad (7.19)$$

Let's now compute the 3 derivatives we need. Since in the computation graph $L = ce$, we can directly compute the derivative $\frac{\partial L}{\partial c}$:

$$\frac{\partial L}{\partial c} = e \quad (7.20)$$

For the other two, we'll need to use the chain rule:

$$\begin{aligned} \frac{\partial L}{\partial a} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial a} \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial d} \frac{\partial d}{\partial b} \end{aligned} \quad (7.21)$$

Eq. 7.21 thus requires five intermediate derivatives: $\frac{\partial L}{\partial e}$, $\frac{\partial L}{\partial c}$, $\frac{\partial e}{\partial a}$, $\frac{\partial e}{\partial d}$, and $\frac{\partial d}{\partial b}$, which are as follows (making use of the fact that the derivative of a sum is the sum of the derivatives):

$$\begin{aligned} L = ce &: \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e \\ e = a + d &: \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1 \\ d = 2b &: \quad \frac{\partial d}{\partial b} = 2 \end{aligned}$$

In the backward pass, we compute each of these partials along each edge of the graph from right to left, multiplying the necessary partials to result in the final derivative we need. Thus we begin by annotating the final node with $\frac{\partial L}{\partial L} = 1$. Moving to the left, we then compute $\frac{\partial L}{\partial c}$ and $\frac{\partial L}{\partial e}$, and so on, until we have annotated the graph all the way to the input variables. The forward pass conveniently already will have computed the values of the forward intermediate variables we need (like d and e) to compute these derivatives. Fig. 7.10 shows the backward pass. At each node we need to compute the local partial derivative with respect to the parent, multiply it by the partial derivative that is being passed down from the parent, and then pass it to the child.

Backward differentiation for a neural network

Of course computation graphs for real neural networks are much more complex. Fig. 7.11 shows a sample computation graph for a 2-layer neural network with $n_0 =$

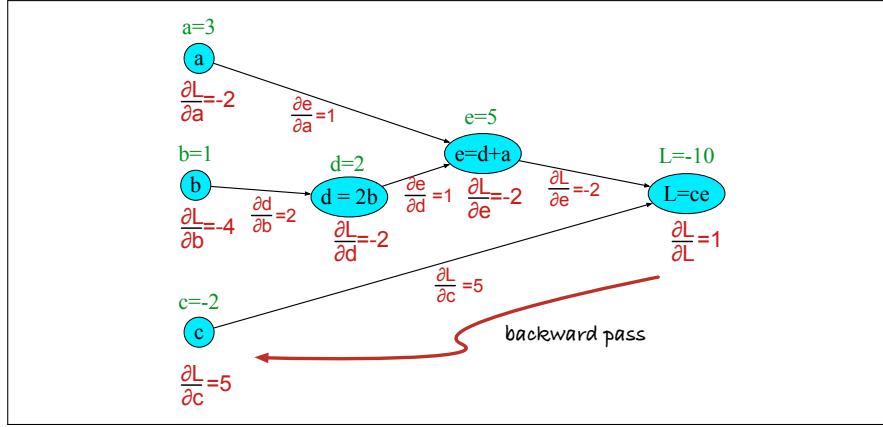


Figure 7.10 Computation graph for the function $L(a, b, c) = c(a + 2b)$, showing the backward pass computation of $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$.

2, $n_1 = 2$, and $n_2 = 1$, assuming binary classification and hence using a sigmoid output unit for simplicity. The function that the computation graph is computing is:

$$\begin{aligned}
 z^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]} \\
 a^{[1]} &= \text{ReLU}(z^{[1]}) \\
 z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
 a^{[2]} &= \sigma(z^{[2]}) \\
 \hat{y} &= a^{[2]}
 \end{aligned} \tag{7.22}$$

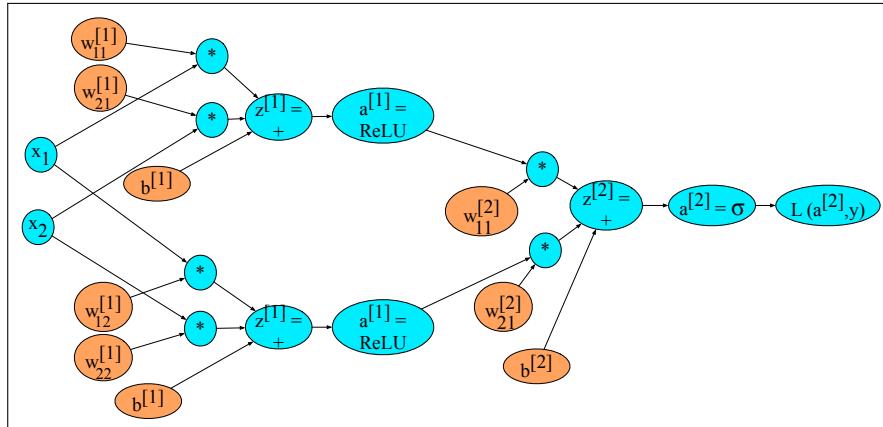


Figure 7.11 Sample computation graph for a simple 2-layer neural net (= 1 hidden layer) with two input dimensions and 2 hidden dimensions.

The weights that need updating (those for which we need to know the partial derivative of the loss function) are shown in orange. In order to do the backward pass, we'll need to know the derivatives of all the functions in the graph. We already saw in Section 5.8 the derivative of the sigmoid σ :

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z)) \tag{7.23}$$

We'll also need the derivatives of each of the other activation functions. The derivative of tanh is:

$$\frac{d \tanh(z)}{dz} = 1 - \tanh^2(z) \quad (7.24)$$

The derivative of the ReLU is

$$\frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases} \quad (7.25)$$

7.4.5 More details on learning

Optimization in neural networks is a non-convex optimization problem, more complex than for logistic regression, and for that and other reasons there are many best practices for successful learning.

For logistic regression we can initialize gradient descent with all the weights and biases having the value 0. In neural networks, by contrast, we need to initialize the weights with small random numbers. It's also helpful to normalize the input values to have 0 mean and unit variance.

Various forms of regularization are used to prevent overfitting. One of the most important is **dropout**: randomly dropping some units and their connections from the network during training (Hinton et al. 2012, Srivastava et al. 2014). Tuning of **hyperparameters** is also important. The parameters of a neural network are the weights W and biases b ; those are learned by gradient descent. The hyperparameters are things that are chosen by the algorithm designer; optimal values are tuned on a devset rather than by gradient descent learning on the training set. Hyperparameters include the learning rate η , the mini-batch size, the model architecture (the number of layers, the number of hidden nodes per layer, the choice of activation functions), how to regularize, and so on. Gradient descent itself also has many architectural variants such as Adam (Kingma and Ba, 2015).

Finally, most modern neural networks are built using computation graph formalisms that make it easy and natural to do gradient computation and parallelization onto vector-based GPUs (Graphic Processing Units). Pytorch (Paszke et al., 2017) and TensorFlow (Abadi et al., 2015) are two of the most popular. The interested reader should consult a neural network textbook for further details; some suggestions are at the end of the chapter.

7.5 Neural Language Models

As our first application of neural networks, let's consider **language modeling**: predicting upcoming words from prior word context.

Neural net-based language models turn out to have many advantages over the n-gram language models of Chapter 3. Among these are that neural language models don't need smoothing, they can handle much longer histories, and they can generalize over contexts of similar words. For a training set of a given size, a neural language model has much higher predictive accuracy than an n-gram language model. Furthermore, neural language models underlie many of the models we'll introduce for tasks like machine translation, dialog, and language generation.

On the other hand, there is a cost for this improved performance: neural net language models are strikingly slower to train than traditional language models, and so for many tasks an n-gram language model is still the right tool.

In this chapter we'll describe simple feedforward neural language models, first introduced by [Bengio et al. \(2003\)](#). Modern neural language models are generally not feedforward but recurrent, using the technology that we will introduce in Chapter 9.

A feedforward neural LM is a standard feedforward network that takes as input at time t a representation of some number of previous words (w_{t-1}, w_{t-2} , etc.) and outputs a probability distribution over possible next words. Thus—like the n-gram LM—the feedforward neural LM approximates the probability of a word given the entire prior context $P(w_t | w_1^{t-1})$ by approximating based on the N previous words:

$$P(w_t | w_1^{t-1}) \approx P(w_t | w_{t-N+1}^{t-1}) \quad (7.26)$$

In the following examples we'll use a 4-gram example, so we'll show a net to estimate the probability $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$.

7.5.1 Embeddings

In neural language models, the prior context is represented by embeddings of the previous words. Representing the prior context as embeddings, rather than by exact words as used in n-gram language models, allows neural language models to generalize to unseen data much better than n-gram language models. For example, suppose we've seen this sentence in training:

I have to make sure when I get home to feed the cat.

but we've never seen the word "dog" after the words "feed the". In our test set we are trying to predict what comes after the prefix "I forgot when I got home to feed the".

An n-gram language model will predict "cat", but not "dog". But a neural LM, which can make use of the fact that "cat" and "dog" have similar embeddings, will be able to assign a reasonably high probability to "dog" as well as "cat", merely because they have similar vectors.

Let's see how this works in practice. Let's assume we have an embedding dictionary E that gives us, for each word in our vocabulary V , the embedding for that word, perhaps precomputed by an algorithm like word2vec from Chapter 6.

[Fig. 7.12](#) shows a sketch of this simplified feedforward neural language model with $N=3$; we have a moving window at time t with an embedding vector representing each of the 3 previous words (words w_{t-1}, w_{t-2} , and w_{t-3}). These 3 vectors are concatenated together to produce x , the input layer of a neural network whose output is a softmax with a probability distribution over words. Thus y_{42} , the value of output node 42 is the probability of the next word w_t being V_{42} , the vocabulary word with index 42.

The model shown in [Fig. 7.12](#) is quite sufficient, assuming we learn the embeddings separately by a method like the **word2vec** methods of Chapter 6. The method of using another algorithm to learn the embedding representations we use for input words is called **pretraining**. If those pretrained embeddings are sufficient for your purposes, then this is all you need.

However, often we'd like to learn the embeddings simultaneously with training the network. This is true when whatever task the network is designed for (sentiment

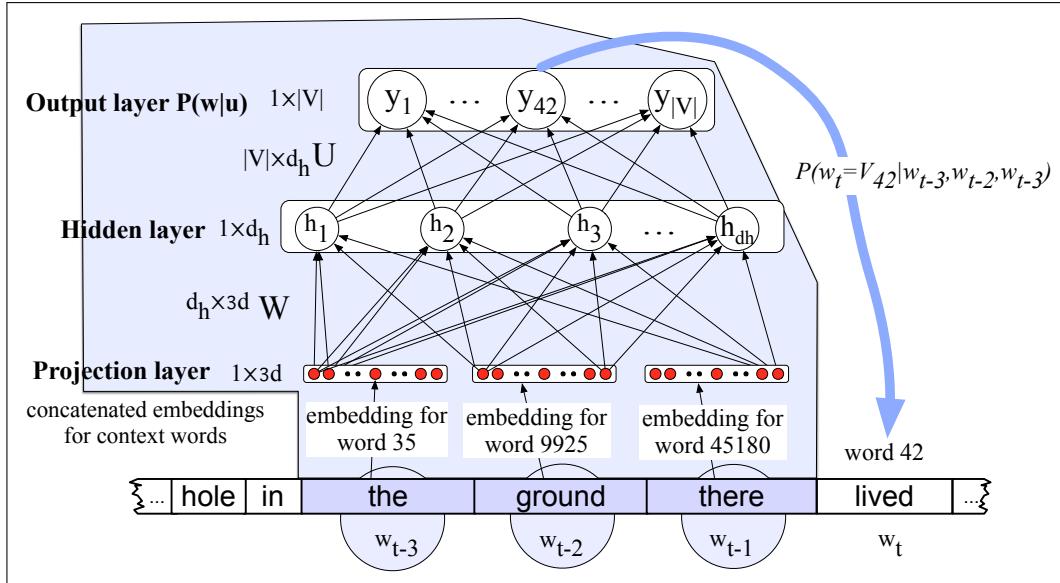


Figure 7.12 A simplified view of a feedforward neural language model moving through a text. At each timestep t the network takes the 3 context words, converts each to a d -dimensional embedding, and concatenates the 3 embeddings together to get the $1 \times Nd$ unit input layer x for the network. These units are multiplied by a weight matrix W and bias vector b and then an activation function to produce a hidden layer h , which is then multiplied by another weight matrix U . (For graphic simplicity we don't show b in this and future pictures.) Finally, a softmax output layer predicts at each node i the probability that the next word w_t will be vocabulary word V_i . (This picture is simplified because it assumes we just look up in an embedding dictionary E the d -dimensional embedding vector for each word, precomputed by an algorithm like word2vec.)

classification, or translation, or parsing) places strong constraints on what makes a good representation.

Let's therefore show an architecture that allows the embeddings to be learned. To do this, we'll add an extra layer to the network, and propagate the error all the way back to the embedding vectors, starting with embeddings with random values and slowly moving toward sensible representations.

one-hot vector

For this to work at the input layer, instead of pre-trained embeddings, we're going to represent each of the N previous words as a one-hot vector of length $|V|$, i.e., with one dimension for each word in the vocabulary. A **one-hot vector** is a vector that has one element equal to 1—in the dimension corresponding to that word's index in the vocabulary—while all the other elements are set to zero.

Thus in a one-hot representation for the word “toothpaste”, supposing it happens to have index 5 in the vocabulary, x_5 is one and $x_i = 0 \ \forall i \neq 5$, as shown here:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ \dots \ \dots \ |V|$$

Fig. 7.13 shows the additional layers needed to learn the embeddings during LM training. Here the $N=3$ context words are represented as 3 one-hot vectors, fully connected to the embedding layer via 3 instantiations of the embedding matrix E . Note that we don't want to learn separate weight matrices for mapping each of the 3 previous words to the projection layer, we want one single embedding dictionary E that's shared among these three. That's because over time, many different words will appear as w_{t-2} or w_{t-1} , and we'd like to just represent each word with one vector, whichever context position it appears in. The embedding weight matrix E thus has

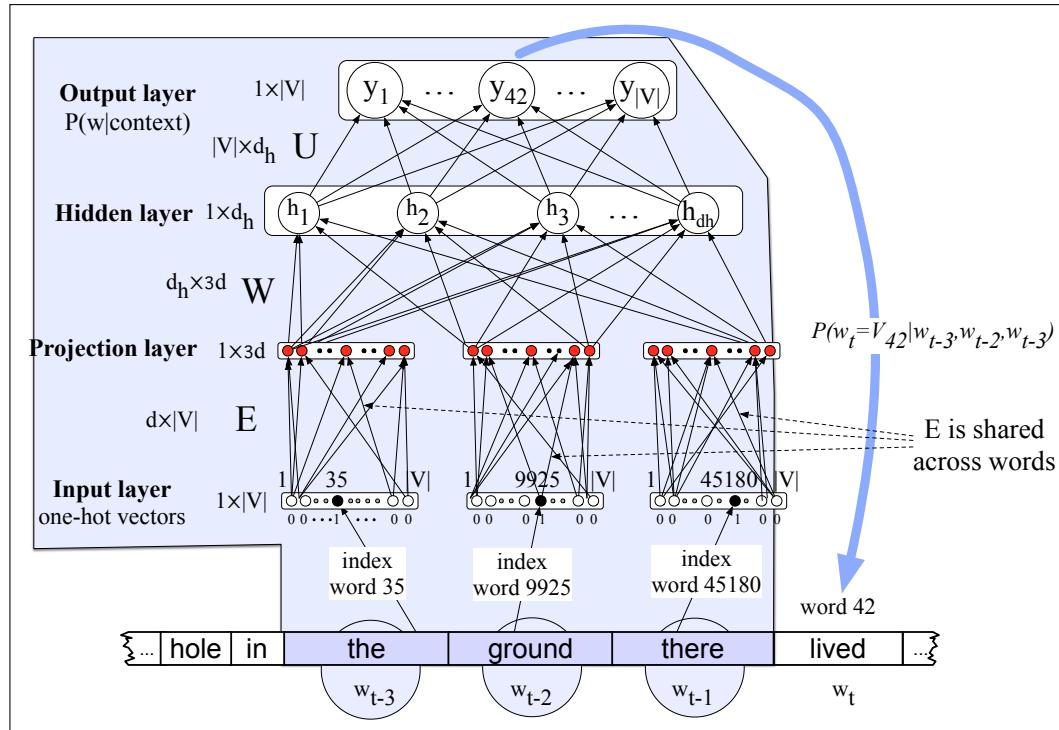


Figure 7.13 Learning all the way back to embeddings. Notice that the embedding matrix E is shared among the 3 context words.

a row for each word, each a vector of d dimensions, and hence has dimensionality $V \times d$.

Let's walk through the forward pass of Fig. 7.13.

projection layer

- Select three embeddings from E :** Given the three previous words, we look up their indices, create 3 one-hot vectors, and then multiply each by the embedding matrix E . Consider w_{t-3} . The one-hot vector for 'the' (index 35) is multiplied by the embedding matrix E , to give the first part of the first hidden layer, called the **projection layer**. Since each row of the input matrix E is just an embedding for a word, and the input is a one-hot column vector x_i for word V_i , the projection layer for input w will be $Ex_i = e_i$, the embedding for word i . We now concatenate the three embeddings for the context words.
- Multiply by W :** We now multiply by W (and add b) and pass through the rectified linear (or other) activation function to get the hidden layer h .
- Multiply by U :** h is now multiplied by U
- Apply softmax:** After the softmax, each node i in the output layer estimates the probability $P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$

In summary, if we use e to represent the projection layer, formed by concatenating the 3 embeddings for the three context vectors, the equations for a neural language model become:

$$e = (Ex_1, Ex_2, \dots, Ex) \quad (7.27)$$

$$h = \sigma(We + b) \quad (7.28)$$

$$z = Uh \quad (7.29)$$

$$y = \text{softmax}(z) \quad (7.30)$$

7.5.2 Training the neural language model

To train the model, i.e. to set all the parameters $\theta = E, W, U, b$, we do gradient descent (Fig. 5.5), using error backpropagation on the computation graph to compute the gradient. Training thus not only sets the weights W and U of the network, but also as we're predicting upcoming words, we're learning the embeddings E for each words that best predict upcoming words.

Generally training proceeds by taking as input a very long text, concatenating all the sentences, starting with random weights, and then iteratively moving through the text predicting each word w_t . At each word w_t , the cross-entropy (negative log likelihood) loss is:

$$L = -\log p(w_t | w_{t-1}, \dots, w_{t-n+1}) \quad (7.31)$$

The gradient for this loss is then:

$$\theta_{t+1} = \theta_t - \eta \frac{\partial -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})}{\partial \theta} \quad (7.32)$$

This gradient can be computed in any standard neural network framework which will then backpropagate through U, W, b, E .

Training the parameters to minimize loss will result both in an algorithm for language modeling (a word predictor) but also a new set of embeddings E that can be used as word representations for other tasks.

7.6 Summary

- Neural networks are built out of **neural units**, originally inspired by human neurons but now simply an abstract computational device.
- Each neural unit multiplies input values by a weight vector, adds a bias, and then applies a non-linear activation function like sigmoid, tanh, or rectified linear.
- In a **fully-connected, feedforward** network, each unit in layer i is connected to each unit in layer $i + 1$, and there are no cycles.
- The power of neural networks comes from the ability of early layers to learn representations that can be utilized by later layers in the network.
- Neural networks are trained by optimization algorithms like **gradient descent**.
- **Error backpropagation**, backward differentiation on a **computation graph**, is used to compute the gradients of the loss function for a network.
- **Neural language models** use a neural network as a probabilistic classifier, to compute the probability of the next word given the previous n words.
- Neural language models can use pretrained **embeddings**, or can learn embeddings from scratch in the process of language modeling.

Bibliographical and Historical Notes

The origins of neural networks lie in the 1940s **McCulloch-Pitts neuron** (McCulloch and Pitts, 1943), a simplified model of the human neuron as a kind of com-

puting element that could be described in terms of propositional logic. By the late 1950s and early 1960s, a number of labs (including Frank Rosenblatt at Cornell and Bernard Widrow at Stanford) developed research into neural networks; this phase saw the development of the perceptron (Rosenblatt, 1958), and the transformation of the threshold into a bias, a notation we still use (Widrow and Hoff, 1960).

The field of neural networks declined after it was shown that a single perceptron unit was unable to model functions as simple as XOR (Minsky and Papert, 1969). While some small amount of work continued during the next two decades, a major revival for the field didn't come until the 1980s, when practical tools for building deeper networks like error backpropagation became widespread (Rumelhart et al., 1986). During the 1980s a wide variety of neural network and related architectures were developed, particularly for applications in psychology and cognitive science (Rumelhart and McClelland 1986b, McClelland and Elman 1986, Rumelhart and McClelland 1986a, Elman 1990), for which the term **connectionist** or **parallel distributed processing** was often used (Feldman and Ballard 1982, Smolensky 1988). Many of the principles and techniques developed in this period are foundational to modern work, including the ideas of distributed representations (Hinton, 1986), recurrent networks (Elman, 1990), and the use of tensors for compositionality (Smolensky, 1990).

By the 1990s larger neural networks began to be applied to many practical language processing tasks as well, like handwriting recognition (LeCun et al. 1989, LeCun et al. 1990) and speech recognition (Morgan and Bourlard 1989, Morgan and Bourlard 1990). By the early 2000s, improvements in computer hardware and advances in optimization and training techniques made it possible to train even larger and deeper networks, leading to the modern term **deep learning** (Hinton et al. 2006, Bengio et al. 2007). We cover more related history in Chapter 9.

There are a number of excellent books on the subject. Goldberg (2017) has a superb and comprehensive coverage of neural networks for natural language processing. For neural networks in general see Goodfellow et al. (2016) and Nielsen (2015).

connectionist

Part-of-Speech Tagging

parts of speech

Dionysius Thrax of Alexandria (c. 100 B.C.), or perhaps someone else (it was a long time ago), wrote a grammatical sketch of Greek (a “*techne*”) that summarized the linguistic knowledge of his day. This work is the source of an astonishing proportion of modern linguistic vocabulary, including words like *syntax*, *diphthong*, *clitic*, and *analogy*. Also included are a description of eight **parts of speech**: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. Although earlier scholars (including Aristotle as well as the Stoics) had their own lists of parts of speech, it was Thrax’s set of eight that became the basis for practically all subsequent part-of-speech descriptions of most European languages for the next 2000 years.

Schoolhouse Rock was a series of popular animated educational television clips from the 1970s. Its Grammar Rock sequence included songs about exactly 8 parts of speech, including the late great Bob Dorough’s *Conjunction Junction*:

*Conjunction Junction, what’s your function?
Hooking up words and phrases and clauses...*

Although the list of 8 was slightly modified from Thrax’s original, the astonishing durability of the parts of speech through two millennia is an indicator of both the importance and the transparency of their role in human language.¹

POS

Parts of speech (also known as **POS**, **word classes**, or **syntactic categories**) are useful because they reveal a lot about a word and its neighbors. Knowing whether a word is a **noun** or a **verb** tells us about likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and syntactic structure (nouns are generally part of noun phrases), making part-of-speech tagging a key aspect of parsing (Chapter 13). Parts of speech are useful features for labeling **named entities** like people or organizations in **information extraction** (Chapter 18), or for coreference resolution (Chapter 22). A word’s part of speech can even play a role in speech recognition or synthesis, e.g., the word *content* is pronounced *COntent* when it is a noun and *conTENT* when it is an adjective.

This chapter introduces parts of speech, and then introduces two algorithms for **part-of-speech tagging**, the task of assigning parts of speech to words. One is generative—Hidden Markov Model (HMM)—and one is discriminative—the Maximum Entropy Markov Model (MEMM). Chapter 9 then introduces a third algorithm based on the recurrent neural network (RNN). All three have roughly equal performance but, as we’ll see, have different tradeoffs.

8.1 (Mostly) English Word Classes

Until now we have been using part-of-speech terms like **noun** and **verb** rather freely. In this section we give a more complete definition of these and other classes. While word classes do have semantic tendencies—adjectives, for example, often describe

¹ Nonetheless, eight isn’t very many and, as we’ll see, recent tagsets have more.

properties and nouns *people*—parts of speech are traditionally defined instead based on syntactic and morphological function, grouping words that have similar neighboring words (their **distributional** properties) or take similar affixes (their morphological properties).

closed class

open class

function word

Parts of speech can be divided into two broad supercategories: **closed class** types and **open class** types. Closed classes are those with relatively fixed membership, such as prepositions—new prepositions are rarely coined. By contrast, nouns and verbs are open classes—new nouns and verbs like *iPhone* or *to fax* are continually being created or borrowed. Any given speaker or corpus may have different open class words, but all speakers of a language, and sufficiently large corpora, likely share the set of closed class words. Closed class words are generally **function words** like *of*, *it*, *and*, or *you*, which tend to be very short, occur frequently, and often have structuring uses in grammar.

noun

Four major open classes occur in the languages of the world: **nouns**, **verbs**, **adjectives**, and **adverbs**. English has all four, although not every language does. The syntactic class **noun** includes the words for most people, places, or things, but others as well. Nouns include concrete terms like *ship* and *chair*, abstractions like *bandwidth* and *relationship*, and verb-like terms like *pacing* as in *His pacing to and fro became quite annoying*. What defines a noun in English, then, are things like its ability to occur with determiners (*a goat*, *its bandwidth*, *Plato's Republic*), to take possessives (*IBM's annual revenue*), and for most but not all nouns to occur in the plural form (*goats*, *abaci*).

proper noun

common noun

count noun

mass noun

verb

adjective

adverb

Open class nouns fall into two classes. **Proper nouns**, like *Regina*, *Colorado*, and *IBM*, are names of specific persons or entities. In English, they generally aren't preceded by articles (e.g., *the book is upstairs*, but *Regina is upstairs*). In written English, proper nouns are usually capitalized. The other class, **common nouns**, are divided in many languages, including English, into **count nouns** and **mass nouns**. Count nouns allow grammatical enumeration, occurring in both the singular and plural (*goat/goats*, *relationship/relationships*) and they can be counted (*one goat*, *two goats*). Mass nouns are used when something is conceptualized as a homogeneous group. So words like *snow*, *salt*, and *communism* are not counted (i.e., **two snows* or **two communisms*). Mass nouns can also appear without articles where singular count nouns cannot (*Snow is white* but not **Goat is white*).

Verbs refer to actions and processes, including main verbs like *draw*, *provide*, and *go*. English verbs have inflections (non-third-person-sg (*eat*), third-person-sg (*eats*), progressive (*eating*), past participle (*eaten*)). While many researchers believe that all human languages have the categories of noun and verb, others have argued that some languages, such as Riau Indonesian and Tongan, don't even make this distinction (Broschart 1997; Evans 2000; Gil 2000).

The third open class English form is **adjectives**, a class that includes many terms for properties or qualities. Most languages have adjectives for the concepts of color (*white*, *black*), age (*old*, *young*), and value (*good*, *bad*), but there are languages without adjectives. In Korean, for example, the words corresponding to English adjectives act as a subclass of verbs, so what is in English an adjective “beautiful” acts in Korean like a verb meaning “to be beautiful”.

The final open class form, **adverbs**, is rather a hodge-podge in both form and meaning. In the following all the italicized words are adverbs:

Actually, I ran home *extremely* quickly yesterday

What coherence the class has semantically may be solely that each of these words can be viewed as modifying something (often verbs, hence the name “ad-

locative adverbs (*home, here, downhill*) specify the direction or location of some action; **degree adverbs** (*extremely, very, somewhat*) specify the extent of some action, process, or property; **manner adverbs** (*slowly, slinkily, delicately*) describe the manner of some action or process; and **temporal adverbs** describe the time that some action or event took place (*yesterday, Monday*). Because of the heterogeneous nature of this class, some adverbs (e.g., temporal adverbs like *Monday*) are tagged in some tagging schemes as nouns.

The closed classes differ more from language to language than do the open classes. Some of the important closed classes in English include:

- prepositions:** on, under, over, near, by, at, from, to, with
- particles:** up, down, on, off, in, out, at, by
- determiners:** a, an, the
- conjunctions:** and, but, or, as, if, when
- pronouns:** she, who, I, others
- auxiliary verbs:** can, may, should, are
- numerals:** one, two, three, first, second, third

preposition **Prepositions** occur before noun phrases. Semantically they often indicate spatial or temporal relations, whether literal (*on it, before then, by the house*) or metaphorical (*on time, with gusto, beside herself*), but often indicate other relations as well, like marking the agent in *Hamlet was written by Shakespeare*. A **particle** resembles a preposition or an adverb and is used in combination with a verb. Particles often have extended meanings that aren't quite the same as the prepositions they resemble, as in the particle *over* in *she turned the paper over*.

phrasal verb A verb and a particle that act as a single syntactic and/or semantic unit are called a **phrasal verb**. The meaning of phrasal verbs is often problematically **non-compositional**—not predictable from the distinct meanings of the verb and the particle. Thus, *turn down* means something like ‘reject’, *rule out* ‘eliminate’, *find out* ‘discover’, and *go on* ‘continue’.

determiner **article** A closed class that occurs with nouns, often marking the beginning of a noun phrase, is the **determiner**. One small subtype of determiners is the **article**: English has three articles: *a, an*, and *the*. Other determiners include *this* and *that* (*this chapter, that page*). *A* and *an* mark a noun phrase as indefinite, while *the* can mark it as definite; definiteness is a discourse property (Chapter 23). Articles are quite frequent in English; indeed, *the* is the most frequently occurring word in most corpora of written English, and *a* and *an* are generally right behind.

conjunctions **Conjunctions** join two phrases, clauses, or sentences. Coordinating conjunctions like *and, or*, and *but* join two elements of equal status. Subordinating conjunctions are used when one of the elements has some embedded status. For example, *that* in “*I thought that you might like some milk*” is a subordinating conjunction that links the main clause *I thought* with the subordinate clause *you might like some milk*. This clause is called subordinate because this entire clause is the “content” of the main verb *thought*. Subordinating conjunctions like *that* which link a verb to its argument in this way are also called **complementizers**.

complementizer **pronoun** **personal possessive** **wh** **Pronouns** are forms that often act as a kind of shorthand for referring to some noun phrase or entity or event. **Personal pronouns** refer to persons or entities (*you, she, I, it, me, etc.*). **Possessive pronouns** are forms of personal pronouns that indicate either actual possession or more often just an abstract relation between the person and some object (*my, your, his, her, its, one's, our, their*). **Wh-pronouns** (*what, who, whom, whoever*) are used in certain question forms, or may also act as

complementizers (*Frida, who married Diego...*).

auxiliary A closed class subtype of English verbs are the **auxiliary** verbs. Cross-linguistically, auxiliaries mark semantic features of a main verb: whether an action takes place in the present, past, or future (tense), whether it is completed (aspect), whether it is negated (polarity), and whether an action is necessary, possible, suggested, or desired (mood). English auxiliaries include the **copula** verb *be*, the two verbs *do* and *have*, along with their inflected forms, as well as a class of **modal verbs**. *Be* is called a copula because it connects subjects with certain kinds of predicate nominals and adjectives (*He is a duck*). The verb *have* can mark the perfect tenses (*I have gone, I had gone*), and *be* is used as part of the passive (*We were robbed*) or progressive (*We are leaving*) constructions. Modals are used to mark the mood associated with the event depicted by the main verb: *can* indicates ability or possibility, *may* permission or possibility, *must* necessity. There is also a modal use of *have* (e.g., *I have to go*).

interjection negative English also has many words of more or less unique function, including **interjections** (*oh, hey, alas, uh, um*), **negatives** (*no, not*), **politeness markers** (*please, thank you*), **greetings** (*Hello, goodbye*), and the existential **there** (*there are two on the table*) among others. These classes may be distinguished or lumped together as interjections or adverbs depending on the purpose of the labeling.

8.2 The Penn Treebank Part-of-Speech Tagset

An important tagset for English is the 45-tag Penn Treebank tagset (Marcus et al., 1993), shown in Fig. 8.1, which has been used to label many corpora. In such labelings, parts of speech are generally represented by placing the tag after each word, delimited by a slash:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	<i>and, but, or</i>	PDT	predeterminer	<i>all, both</i>	VBP	verb non-3sg present	<i>eat</i>
CD	cardinal number	<i>one, two</i>	POS	possessive ending	's	VBZ	verb 3sg pres	<i>eats</i>
DT	determiner	<i>a, the</i>	PRP	personal pronoun	<i>I, you, he</i>	WDT	wh-determ.	<i>which, that</i>
EX	existential ‘there’	<i>there</i>	PRP\$	possess. pronoun	<i>your, one's</i>	WP	wh-pronoun	<i>what, who</i>
FW	foreign word	<i>mea culpa</i>	RB	adverb	<i>quickly</i>	WP\$	wh-possess.	<i>whose</i>
IN	preposition/subordin-conj	<i>of, in, by</i>	RBR	comparative	<i>faster</i>	WRB	wh-adverb	<i>how, where</i>
JJ	adjective	<i>yellow</i>	RBS	superlatv. adverb	<i>fastest</i>	\$	dollar sign	\$
JJR	comparative adj	<i>bigger</i>	RP	particle	<i>up, off</i>	#	pound sign	#
JJS	superlative adj	<i>wildest</i>	SYM	symbol	+%, &	"	left quote	‘ or “
LS	list item marker	<i>1, 2, One</i>	TO	“to”	<i>to</i>	"	right quote	’ or ”
MD	modal	<i>can, should</i>	UH	interjection	<i>ah, oops</i>	(left paren	[, (, {, <
NN	sing or mass noun	<i>llama</i>	VB	verb base form	<i>eat</i>)	right paren],), }, >
NNS	noun, plural	<i>llamas</i>	VBD	verb past tense	<i>ate</i>	,	comma	,
NNP	proper noun, sing.	<i>IBM</i>	VBG	verb gerund	<i>eating</i>	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	<i>Carolinas</i>	VBN	verb past part.	<i>eaten</i>	:	sent-mid punc	: ; ... - -

Figure 8.1 Penn Treebank part-of-speech tags (including punctuation).

(8.1) The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

(8.2) There/EX are/VBP 70/CD children/NNS there/RB

- (8.3) Preliminary/JJ findings/NNS were/VBD **reported/VBN** in/IN today/NN
 's/POS New/NNP England/NNP Journal/NNP of/IN Medicine/NNP ./.

Example (8.1) shows the determiners *the* and *a*, the adjectives *grand* and *other*, the common nouns *jury*, *number*, and *topics*, and the past tense verb *commented*. Example (8.2) shows the use of the EX tag to mark the existential *there* construction in English, and, for comparison, another use of *there* which is tagged as an adverb (RB). Example (8.3) shows the segmentation of the possessive morpheme 's, and a passive construction, 'were reported', in which *reported* is tagged as a past participle (VBN). Note that since *New England Journal of Medicine* is a proper noun, the Treebank tagging chooses to mark each noun in it separately as NNP, including *journal* and *medicine*, which might otherwise be labeled as common nouns (NN).

Brown Corpora labeled with parts of speech are crucial training (and testing) sets for statistical tagging algorithms. Three main tagged corpora are consistently used for training and testing part-of-speech taggers for English. The **Brown** corpus is a million words of samples from 500 written texts from different genres published in the United States in 1961. The **WSJ** corpus contains a million words published in the Wall Street Journal in 1989. The **Switchboard** corpus consists of 2 million words of telephone conversations collected in 1990-1991. The corpora were created by running an automatic part-of-speech tagger on the texts and then human annotators hand-corrected each tag.

WSJ

Switchboard

There are some minor differences in the tagsets used by the corpora. For example in the WSJ and Brown corpora, the single Penn tag TO is used for both the infinitive *to* (*I like to race*) and the preposition *to* (*go to the store*), while in Switchboard the tag TO is reserved for the infinitive use of *to* and the preposition is tagged IN:

Well/UH ./, I/PRP ./, I/PRP want/VBP **to/TO** go/VB **to/IN** a/DT restauran-t/NN

Finally, there are some idiosyncracies inherent in any tagset. For example, because the Penn 45 tags were collapsed from a larger 87-tag tagset, the **original Brown tagset**, some potentially useful distinctions were lost. The Penn tagset was designed for a treebank in which sentences were parsed, and so it leaves off syntactic information recoverable from the parse tree. Thus for example the Penn tag IN is used for both subordinating conjunctions like *if*, *when*, *unless*, *after*:

after/IN spending/VBG a/DT day/NN at/IN the/DT beach/NN

and prepositions like *in*, *on*, *after*:

after/IN sunrise/NN

Words are generally tokenized before tagging. The Penn Treebank and the British National Corpus split contractions and the 's-genitive from their stems.²

would/MD n't/RB
 children/NNS 's/POS

The Treebank tagset assumes that tokenization of multipart words like *New York* is done at whitespace, thus tagging. *a New York City firm* as *a/DT New/NNP York/NNP City/NNP firm/NN*.

Another commonly used tagset, the Universal POS tag set of the Universal Dependencies project (Nivre et al., 2016a), is used when building systems that can tag many languages. See Section 8.7.

² Indeed, the Treebank tag POS is used only for 's, which must be segmented in tokenization.

8.3 Part-of-Speech Tagging

part-of-speech tagging

Part-of-speech tagging is the process of assigning a part-of-speech marker to each word in an input text.³ The input to a tagging algorithm is a sequence of (tokenized) words and a tagset, and the output is a sequence of tags, one per token.

ambiguous

Tagging is a **disambiguation** task; words are **ambiguous**—have more than one possible part-of-speech—and the goal is to find the correct tag for the situation. For example, *book* can be a verb (*book that flight*) or a noun (*hand me that book*). *That* can be a determiner (*Does that flight serve dinner*) or a complementizer (*I thought that your flight was earlier*). The goal of POS-tagging is to **resolve** these ambiguities, choosing the proper tag for the context. How common is tag ambiguity? Fig. 8.2 shows that most word types (85-86%) are unambiguous (*Janet* is always NNP, *funniest* JJS, and *hesitantly* RB). But the ambiguous words, though accounting for only 14-15% of the vocabulary, are very common words, and hence 55-67% of word tokens in running text are ambiguous.⁴

		WSJ	Brown
Types:			
Unambiguous	(1 tag)	44,432 (86%)	45,799 (85%)
Ambiguous	(2+ tags)	7,025 (14%)	8,050 (15%)
Tokens:			
Unambiguous	(1 tag)	577,421 (45%)	384,349 (33%)
Ambiguous	(2+ tags)	711,780 (55%)	786,646 (67%)

Figure 8.2 Tag ambiguity for word types in Brown and WSJ, using Treebank-3 (45-tag) tagging. Punctuation were treated as words, and words were kept in their original case.

Some of the most ambiguous frequent words are *that*, *back*, *down*, *put* and *set*; here are some examples of the 6 different parts of speech for the word *back*:

earnings growth took a **back/JJ** seat
 a small building in the **back/NN**
 a clear majority of senators **back/VBP** the bill
 Dave began to **back/VB** toward the door
 enable the country to buy **back/RP** about debt
 I was twenty-one **back/RB** then

Nonetheless, many words are easy to disambiguate, because their different tags aren't equally likely. For example, *a* can be a determiner or the letter *a*, but the determiner sense is much more likely. This idea suggests a simplistic **baseline** algorithm for part-of-speech tagging: given an ambiguous word, choose the tag which is **most frequent** in the training corpus. This is a key concept:

Most Frequent Class Baseline: Always compare a classifier against a baseline at least as good as the most frequent class baseline (assigning each token to the class it occurred in most often in the training set).

accuracy

How good is this baseline? A standard way to measure the performance of part-of-speech taggers is **accuracy**: the percentage of tags correctly labeled (matching

³ Tags are also applied to punctuation, so tagging assumes tokenizing of commas, quotation marks, etc., and disambiguating end-of-sentence periods from periods inside words (*e.g.*, *etc.*).

⁴ Note the large differences across the two genres, especially in token frequency. Tags in the WSJ corpus are less ambiguous; its focus on financial news leads to a more limited distribution of word usages than the diverse genres of the Brown corpus.

human labels on a test set). If we train on the WSJ training corpus and test on sections 22-24 of the same corpus the most-frequent-tag baseline achieves an accuracy of 92.34%. By contrast, the state of the art in part-of-speech tagging on this dataset is around 97% tag accuracy, a performance that is achievable by most algorithms (HMMs, MEMMs, neural networks, rule-based algorithms). See Section 8.7 on other languages and genres.

8.4 HMM Part-of-Speech Tagging

sequence model In this section we introduce the use of the Hidden Markov Model for part-of-speech tagging. The HMM is a **sequence model**. A sequence model or **sequence classifier** is a model whose job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels. An HMM is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences, whatever), it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

8.4.1 Markov Chains

Markov chain

The HMM is based on augmenting the Markov chain. A **Markov chain** is a model that tells us something about the probabilities of sequences of random variables, *states*, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, for example the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. All the states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.

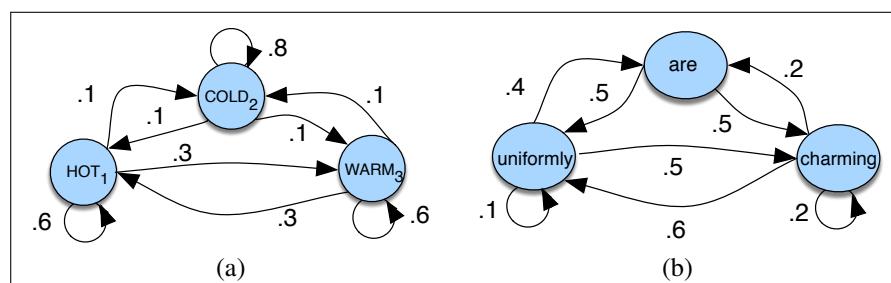


Figure 8.3 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Markov assumption

More formally, consider a sequence of state variables q_1, q_2, \dots, q_i . A Markov model embodies the **Markov assumption** on the probabilities of this sequence: that when predicting the future, the past doesn't matter, only the present.

$$\text{Markov Assumption: } P(q_i = a | q_1 \dots q_{i-1}) = P(q_i = a | q_{i-1}) \quad (8.4)$$

Figure 8.3a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. The

states are represented as nodes in the graph, and the transitions, with their probabilities, as edges. The transitions are probabilities: the values of arcs leaving a given state must sum to 1. Figure 8.3b shows a Markov chain for assigning a probability to a sequence of words $w_1 \dots w_n$. This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability $p(w_i|w_j)!$ Given the two models in Fig. 8.3, we can assign a probability to any sequence from our vocabulary.

Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N states
$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$	a transition probability matrix A , each a_{ij} representing the probability of moving from state i to state j , s.t. $\sum_{j=1}^n a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

Before you go on, use the sample probabilities in Fig. 8.3a (with $\pi = [0.1, 0.7, 0.2]$) to compute the probability of each of the following sequences:

- (8.5) hot hot hot hot
- (8.6) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. 8.3a?

8.4.2 The Hidden Markov Model

hidden

Hidden
Markov model

A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags **hidden** because they are not observed.

A **hidden Markov model (HMM)** allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model. An HMM is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N states
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a transition probability matrix A , each a_{ij} representing the probability of moving from state i to state j , s.t. $\sum_{j=1}^N a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T observations , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of observation likelihoods , also called emission probabilities , each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends only on the previous state:

$$\text{Markov Assumption: } P(q_i|q_1 \dots q_{i-1}) = P(q_i|q_{i-1}) \quad (8.7)$$

Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:

$$\text{Output Independence: } P(o_i|q_1 \dots q_i, \dots, q_T, o_1, \dots, o_i, \dots, o_T) = P(o_i|q_i) \quad (8.8)$$

8.4.3 The components of an HMM tagger

Let's start by looking at the pieces of an HMM tagger, and then we'll see how to use it to tag. An HMM has two components, the A and B probabilities.

The A matrix contains the tag transition probabilities $P(t_i|t_{i-1})$ which represent the probability of a tag occurring given the previous tag. For example, modal verbs like *will* are very likely to be followed by a verb in the base form, a VB, like *race*, so we expect this probability to be high. We compute the maximum likelihood estimate of this transition probability by counting, out of the times we see the first tag in a labeled corpus, how often the first tag is followed by the second:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \quad (8.9)$$

In the WSJ corpus, for example, MD occurs 13124 times of which it is followed by VB 10471, for an MLE estimate of

$$P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80 \quad (8.10)$$

Let's walk through an example, seeing how these probabilities are estimated and used in a sample tagging task, before we return to the algorithm for decoding.

In HMM tagging, the probabilities are estimated by counting on a tagged training corpus. For this example we'll use the tagged WSJ corpus.

The B emission probabilities, $P(w_i|t_i)$, represent the probability, given a tag (say MD), that it will be associated with a given word (say *will*). The MLE of the emission probability is

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \quad (8.11)$$

Of the 13124 occurrences of MD in the WSJ corpus, it is associated with *will* 4046 times:

$$P(will|MD) = \frac{C(MD, will)}{C(MD)} = \frac{4046}{13124} = .31 \quad (8.12)$$

We saw this kind of Bayesian modeling in Chapter 4; recall that this likelihood term is not asking “which is the most likely tag for the word *will*?”. That would be the posterior $P(MD|will)$. Instead, $P(will|MD)$ answers the slightly counterintuitive question “If we were going to generate a MD, how likely is it that this modal would be *will*?“

The A transition probabilities, and B observation likelihoods of the HMM are illustrated in Fig. 8.4 for three states in an HMM part-of-speech tagger; the full tagger would have one state for each tag.



Figure 8.4 An illustration of the two parts of an HMM representation: the A transition probabilities used to compute the prior probability, and the B observation likelihoods that are associated with each state, one likelihood for each possible observation word.

8.4.4 HMM tagging as decoding

For any model, such as an HMM, that contains hidden variables, the task of determining the hidden variables sequence corresponding to the sequence of observations is called **decoding**. More formally,

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \dots, o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \dots q_T$.

For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence t_1^n that is most probable given the observation sequence of n words w_1^n :

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \quad (8.13)$$

The way we'll do this in the HMM is to use Bayes' rule to instead compute:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} \quad (8.14)$$

Furthermore, we simplify Eq. 8.14 by dropping the denominator $P(w_1^n)$:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n) \quad (8.15)$$

HMM taggers make two further simplifying assumptions. The first is that the probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i | t_i) \quad (8.16)$$

The second assumption, the **bigram** assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence;

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1}) \quad (8.17)$$

Plugging the simplifying assumptions from Eq. 8.16 and Eq. 8.17 into Eq. 8.15 results in the following equation for the most probable tag sequence from a bigram tagger:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^n \overbrace{P(w_i | t_i)}^{\text{emission transition}} \overbrace{P(t_i | t_{i-1})}^{\text{transition probability}} \quad (8.18)$$

The two parts of Eq. 8.18 correspond neatly to the **B emission probability** and **A transition probability** that we just defined above!

8.4.5 The Viterbi Algorithm

Viterbi algorithm

The decoding algorithm for HMMs is the **Viterbi algorithm** shown in Fig. 8.5. As an instance of **dynamic programming**, Viterbi resembles the dynamic programming **minimum edit distance** algorithm of Chapter 2.

```

function VITERBI(observations of len  $T$ , state-graph of len  $N$ ) returns best-path, path-prob
    create a path probability matrix viterbi[ $N, T$ ]
    for each state  $s$  from 1 to  $N$  do ; initialization step
        viterbi[ $s, 1$ ]  $\leftarrow \pi_s * b_s(o_1)$ 
        backpointer[ $s, 1$ ]  $\leftarrow 0$ 
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
             $viterbi[s, t] \leftarrow \max_{s'=1}^N viterbi[s', t-1] * a_{s', s} * b_s(o_t)$ 
            backpointer[ $s, t$ ]  $\leftarrow \operatorname{argmax}_{s'=1}^N viterbi[s', t-1] * a_{s', s} * b_s(o_t)$ 
    bestpathprob  $\leftarrow \max_{s=1}^N viterbi[s, T]$  ; termination step
    bestpathpointer  $\leftarrow \operatorname{argmax}_{s=1}^N viterbi[s, T]$  ; termination step
    bestpath  $\leftarrow$  the path starting at state bestpathpointer, that follows backpointer[] to states back in time
    return bestpath, bestpathprob

```

Figure 8.5 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A, B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

The Viterbi algorithm first sets up a probability matrix or **lattice**, with one column for each observation o_t and one row for each state in the state graph. Each column thus has a cell for each state q_i in the single combined automaton. Figure 8.6 shows an intuition of this lattice for the sentence *Janet will back the bill*.

Each cell of the lattice, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence q_1, \dots, q_{t-1} , given the HMM λ . The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability

$$v_t(j) = \max_{q_1, \dots, q_{t-1}} P(q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda) \quad (8.19)$$

We represent the most probable path by taking the maximum over all possible previous state sequences $\max_{q_1, \dots, q_{t-1}}$. Like other dynamic programming algorithms,



Figure 8.6 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t - 1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t , the value $v_t(j)$ is computed as

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) \quad (8.20)$$

The three factors that are multiplied in Eq. 8.20 for extending the previous paths to compute the Viterbi probability at time t are

$v_{t-1}(i)$	the previous Viterbi path probability from the previous time step
a_{ij}	the transition probability from previous state q_i to current state q_j
$b_j(o_t)$	the state observation likelihood of the observation symbol o_t given the current state j

8.4.6 Working through an example

Let's tag the sentence *Janet will back the bill*; the goal is the correct series of tags (see also Fig. 8.6):

(8.21) Janet/NNP will/MD back/VB the/DT bill/NN

Let the HMM be defined by the two tables in Fig. 8.7 and Fig. 8.8. Figure 8.7 lists the a_{ij} probabilities for transitioning between the hidden states (part-of-speech tags). Figure 8.8 expresses the $b_i(o_t)$ probabilities, the *observation likelihood* of words given tags. This table is (slightly simplified) from counts in the WSJ corpus. So the word *Janet* only appears as an NNP, *back* has 4 possible parts of speech, and the word *the* can appear as a determiner or as an NNP (in titles like “Somewhere Over the Rainbow” all words are tagged as NNP).

Figure 8.9 shows a fleshed-out version of the sketch we saw in Fig. 8.6, the Viterbi lattice for computing the best hidden state sequence for the observation sequence *Janet will back the bill*.

	NNP	MD	VB	JJ	NN	RB	DT
<s>	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.7 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.8 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

There are $N = 5$ state columns. We begin in column 1 (for the word *Janet*) by setting the Viterbi value in each cell to the product of the π transition probability (the start probability for that state i , which we get from the $<s>$ entry of Fig. 8.7), and the observation likelihood of the word *Janet* given the tag for that cell. Most of the cells in the column are zero since the word *Janet* cannot be any of those tags. The reader should find this in Fig. 8.9.

Next, each cell in the *will* column gets updated. For each state, we compute the value $viterbi[s, t]$ by taking the maximum over the extensions of all the paths from the previous column that lead to the current cell according to Eq. 8.20. We have shown the values for the MD, VB, and NN cells. Each cell gets the max of the 7 values from the previous column, multiplied by the appropriate transition probability; as it happens in this case, most of them are zero from the previous column. The remaining value is multiplied by the relevant observation probability, and the (trivial) max is taken. In this case the final value, 2.772e-8, comes from the NNP state at the previous column. The reader should fill in the rest of the lattice in Fig. 8.9 and backtrace to see whether or not the Viterbi algorithm returns the gold state sequence NNP MD VB DT NN.

8.4.7 Extending the HMM Algorithm to Trigrams

Practical HMM taggers have a number of extensions of this simple model. One important missing feature is a wider tag context. In the tagger described above the probability of a tag depends only on the previous tag:

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i|t_{i-1}) \quad (8.22)$$



Figure 8.9 The first few entries in the individual state columns for the Viterbi algorithm. Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path. We have only filled out columns 1 and 2; to avoid clutter most cells with value 0 are left empty. The rest is left as an exercise for the reader. After the cells are filled in, backtracking from the *end* state, we should be able to reconstruct the correct state sequence NNP MD VB DT NN.

In practice we use more of the history, letting the probability of a tag depend on the two previous tags:

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1}, t_{i-2}) \quad (8.23)$$

Extending the algorithm from bigram to trigram taggers gives a small (perhaps a half point) increase in performance, but conditioning on two previous tags instead of one requires a significant change to the Viterbi algorithm. For each cell, instead of taking a max over transitions from each cell in the previous column, we have to take a max over paths through the cells in the previous two columns, thus considering N^2 rather than N hidden states at every observation.

In addition to increasing the context window, HMM taggers have a number of other advanced features. One is to let the tagger know the location of the end of the sentence by adding dependence on an end-of-sequence marker for t_{n+1} . This gives the following equation for part-of-speech tagging:

$$\hat{t}_1^n = \operatorname{argmax}_{t_1^n} P(t_1^n | w_1^n) \approx \operatorname{argmax}_{t_1^n} \left[\prod_{i=1}^n P(w_i | t_i) P(t_i | t_{i-1}, t_{i-2}) \right] P(t_{n+1} | t_n) \quad (8.24)$$

In tagging any sentence with Eq. 8.24, three of the tags used in the context will fall off the edge of the sentence, and hence will not match regular words. These tags, t_{-1} , t_0 , and t_{n+1} , can all be set to be a single special ‘sentence boundary’ tag that is added to the tagset, which assumes sentences boundaries have already been marked.

One problem with trigram taggers as instantiated in Eq. 8.24 is data sparsity. Any particular sequence of tags t_{i-2}, t_{i-1}, t_i that occurs in the test set may simply never have occurred in the training set. That means we cannot compute the tag trigram probability just by the maximum likelihood estimate from counts, following Eq. 8.25:

$$P(t_i|t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})} \quad (8.25)$$

Just as we saw with language modeling, many of these counts will be zero in any training set, and we will incorrectly predict that a given tag sequence will never occur! What we need is a way to estimate $P(t_i|t_{i-1}, t_{i-2})$ even if the sequence t_{i-2}, t_{i-1}, t_i never occurs in the training data.

The standard approach to solving this problem is the same interpolation idea we saw in language modeling: estimate the probability by combining more robust, but weaker estimators. For example, if we’ve never seen the tag sequence PRP VB TO, and so can’t compute $P(\text{TO}|\text{PRP}, \text{VB})$ from this frequency, we still could rely on the bigram probability $P(\text{TO}|\text{VB})$, or even the unigram probability $P(\text{TO})$. The maximum likelihood estimation of each of these probabilities can be computed from a corpus with the following counts:

$$\text{Trigrams } \hat{P}(t_i|t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})} \quad (8.26)$$

$$\text{Bigrams } \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \quad (8.27)$$

$$\text{Unigrams } \hat{P}(t_i) = \frac{C(t_i)}{N} \quad (8.28)$$

The standard way to combine these three estimators to estimate the trigram probability $P(t_i|t_{i-1}, t_{i-2})$ is via linear interpolation. We estimate the probability $P(t_i|t_{i-1}t_{i-2})$ by a weighted sum of the unigram, bigram, and trigram probabilities:

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_3 \hat{P}(t_i|t_{i-1}t_{i-2}) + \lambda_2 \hat{P}(t_i|t_{i-1}) + \lambda_1 \hat{P}(t_i) \quad (8.29)$$

deleted
interpolation

We require $\lambda_1 + \lambda_2 + \lambda_3 = 1$, ensuring that the resulting P is a probability distribution. The λ s are set by **deleted interpolation** (Jelinek and Mercer, 1980): we successively delete each trigram from the training corpus and choose the λ s so as to maximize the likelihood of the rest of the corpus. The deletion helps to set the λ s in such a way as to generalize to unseen data and not overfit. Figure 8.10 gives a deleted interpolation algorithm for tag trigrams.

8.4.8 Beam Search

When the number of states grows very large, the vanilla Viterbi algorithm is slow. The complexity of the algorithm is $O(N^2T)$; N (the number of states) can be large for trigram taggers, which have to consider every previous pair of the 45 tags, resulting in $45^3 = 91,125$ computations per column. N can be even larger for other applications of Viterbi, for example to decoding in neural networks, as we will see in future chapters.

```

function DELETED-INTERPOLATION(corpus) returns  $\lambda_1, \lambda_2, \lambda_3$ 
   $\lambda_1, \lambda_2, \lambda_3 \leftarrow 0$ 
  foreach trigram  $t_1, t_2, t_3$  with  $C(t_1, t_2, t_3) > 0$ 
    depending on the maximum of the following three values
      case  $\frac{C(t_1, t_2, t_3) - 1}{C(t_1, t_2) - 1}$ : increment  $\lambda_3$  by  $C(t_1, t_2, t_3)$ 
      case  $\frac{C(t_2, t_3) - 1}{C(t_2) - 1}$ : increment  $\lambda_2$  by  $C(t_1, t_2, t_3)$ 
      case  $\frac{C(t_3) - 1}{N - 1}$ : increment  $\lambda_1$  by  $C(t_1, t_2, t_3)$ 
    end
  end
  normalize  $\lambda_1, \lambda_2, \lambda_3$ 
  return  $\lambda_1, \lambda_2, \lambda_3$ 

```

Figure 8.10 The deleted interpolation algorithm for setting the weights for combining unigram, bigram, and trigram tag probabilities. If the denominator is 0 for any case, we define the result of that case to be 0. N is the number of tokens in the corpus. After Brants (2000).

beam search

One common solution to the complexity problem is the use of **beam search** decoding. In beam search, instead of keeping the entire column of states at each time point t , we just keep the best few hypothesis at that point. At time t this requires computing the Viterbi score for each of the N cells, sorting the scores, and keeping only the best-scoring states. The rest are pruned out and not continued forward to time $t + 1$.

beam width

One way to implement beam search is to keep a fixed number of states instead of all N current states. Here the **beam width** β is a fixed number of states. Alternatively β can be modeled as a fixed percentage of the N states, or as a probability threshold. Figure 8.11 shows the search lattice using a beam width of 2 states.

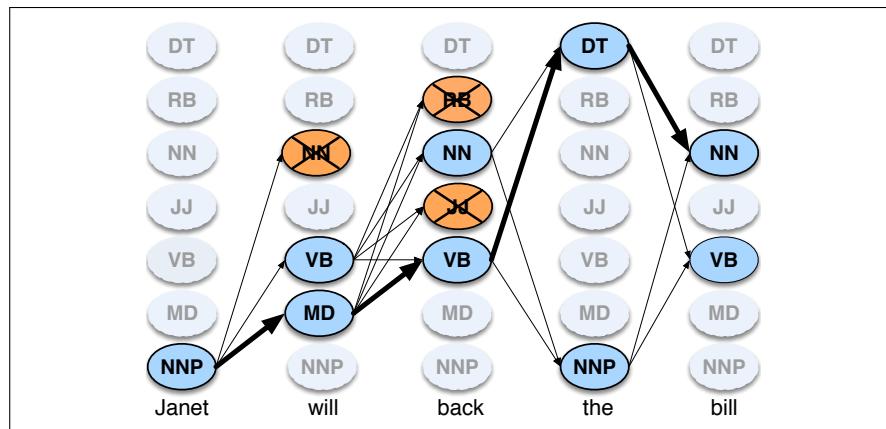


Figure 8.11 A beam search version of Fig. 8.6, showing a beam width of 2. At each time t , all (non-zero) states are computed, but then they are sorted and only the best 2 states are propagated forward and the rest are pruned, shown in orange.

8.4.9 Unknown Words

*words people
never use —
could be
only I
know them*

Ishikawa Takuboku 1885–1912

unknown words

To achieve high accuracy with part-of-speech taggers, it is also important to have a good model for dealing with **unknown words**. Proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. One useful feature for distinguishing parts of speech is word shape: words starting with capital letters are likely to be proper nouns (NNP).

But the strongest source of information for guessing the part-of-speech of unknown words is morphology. Words that end in *-s* are likely to be plural nouns (NNS), words ending with *-ed* tend to be past participles (VBN), words ending with *-able* adjectives (JJ), and so on. We store for each final letter sequence (for simplicity referred to as word *suffixes*) of up to 10 letters the statistics of the tag it was associated with in training. We are thus computing for each suffix of length i the probability of the tag t_i given the suffix letters (Samuelsson 1993, Brants 2000):

$$P(t_i | l_{n-i+1} \dots l_n) \quad (8.30)$$

Back-off is used to smooth these probabilities with successively shorter suffixes. Because unknown words are unlikely to be closed-class words like prepositions, suffix probabilities can be computed only for words whose training set frequency is ≤ 10 , or only for open-class words. Separate suffix tries are kept for capitalized and uncapitalized words.

Finally, because Eq. 8.30 gives a posterior estimate $p(t_i | w_i)$, we can compute the likelihood $p(w_i | t_i)$ that HMMs require by using Bayesian inversion (i.e., using Bayes' rule and computation of the two priors $P(t_i)$ and $P(t_i | l_{n-i+1} \dots l_n)$).

In addition to using capitalization information for unknown words, Brants (2000) also uses capitalization for known words by adding a capitalization feature to each tag. Thus, instead of computing $P(t_i | t_{i-1}, t_{i-2})$ as in Eq. 8.26, the algorithm computes the probability $P(t_i, c_i | t_{i-1}, c_{i-1}, t_{i-2}, c_{i-2})$. This is equivalent to having a capitalized and uncapitalized version of each tag, doubling the size of the tagset.

Combining all these features, a trigram HMM like that of Brants (2000) has a tagging accuracy of 96.7% on the Penn Treebank, perhaps just slightly below the performance of the best MEMM and neural taggers.

8.5 Maximum Entropy Markov Models

While an HMM can achieve very high accuracy, we saw that it requires a number of architectural innovations to deal with unknown words, backoff, suffixes, and so on. It would be so much easier if we could add arbitrary features directly into the model in a clean way, but that's hard for generative models like HMMs. Luckily, we've already seen a model for doing this: the logistic regression model of Chapter 5! But logistic regression isn't a sequence model; it assigns a class to a single observation. However, we could turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word

as a feature in the classification of the next word. When we apply logistic regression in this way, it's called the **maximum entropy Markov model** or **MEMM**⁵

Let the sequence of words be $W = w_1^n$ and the sequence of tags $T = t_1^n$. In an HMM to compute the best tag sequence that maximizes $P(T|W)$ we rely on Bayes' rule and the likelihood $P(W|T)$:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T P(W|T)P(T) \\ &= \operatorname{argmax}_T \prod_i P(\text{word}_i|\text{tag}_i) \prod_i P(\text{tag}_i|\text{tag}_{i-1})\end{aligned}\quad (8.31)$$

In an MEMM, by contrast, we compute the posterior $P(T|W)$ directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned}\quad (8.32)$$

Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability $P(t_i|w_i, t_{i-1})$ in a different way than an HMM. Fig. 8.12 shows the intuition of the difference via the direction of the arrows; HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).



Figure 8.12 A schematic view of the HMM (top) and MEMM (bottom) representation of the probability computation for the correct sequence of tags for the *back* sentence. The HMM computes the likelihood of the observation given the hidden state, while the MEMM computes the posterior of each state, conditioned on the previous state and current observation.

8.5.1 Features in a MEMM

Of course we don't build MEMMs that condition just on w_i and t_{i-1} . The reason to use a discriminative sequence model is that it's easier to incorporate a lot of features.⁶ Figure 8.13 shows a graphical intuition of some of these additional features.

⁵ ‘Maximum entropy model’ is an outdated name for logistic regression; see the history section.

⁶ Because in HMMs all computation is based on the two probabilities $P(\text{tag}|\text{tag})$ and $P(\text{word}|\text{tag})$, if we want to include some source of knowledge into the tagging process, we must find a way to encode the knowledge into one of these two probabilities. Each time we add a feature we have to do a lot of complicated conditioning which gets harder and harder as we have more and more such features.



Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

A basic MEMM part-of-speech tagger conditions on the observation word itself, neighboring words, and previous tags, and various combinations, using feature **templates** like the following:

$$\begin{aligned} \langle t_i, w_{i-2} \rangle, \langle t_i, w_{i-1} \rangle, \langle t_i, w_i \rangle, \langle t_i, w_{i+1} \rangle, \langle t_i, w_{i+2} \rangle \\ \langle t_i, t_{i-1} \rangle, \langle t_i, t_{i-2}, t_{i-1} \rangle, \\ \langle t_i, t_{i-1}, w_i \rangle, \langle t_i, w_{i-1}, w_i \rangle \langle t_i, w_i, w_{i+1} \rangle, \end{aligned} \quad (8.33)$$

Recall from Chapter 5 that feature templates are used to automatically populate the set of features from every instance in the training and test set. Thus our example *Janet/NNP will/MD back/VB the/DT bill/NN*, when w_i is the word *back*, would generate the following features:

- $t_i = \text{VB}$ and $w_{i-2} = \text{Janet}$
- $t_i = \text{VB}$ and $w_{i-1} = \text{will}$
- $t_i = \text{VB}$ and $w_i = \text{back}$
- $t_i = \text{VB}$ and $w_{i+1} = \text{the}$
- $t_i = \text{VB}$ and $w_{i+2} = \text{bill}$
- $t_i = \text{VB}$ and $t_{i-1} = \text{MD}$
- $t_i = \text{VB}$ and $t_{i-1} = \text{MD}$ and $t_{i-2} = \text{NNP}$
- $t_i = \text{VB}$ and $w_i = \text{back}$ and $w_{i+1} = \text{the}$

Also necessary are features to deal with unknown words, expressing properties of the word's spelling or shape:

- w_i contains a particular prefix (from all prefixes of length ≤ 4)
- w_i contains a particular suffix (from all suffixes of length ≤ 4)
- w_i contains a number
- w_i contains an upper-case letter
- w_i contains a hyphen
- w_i is all upper case
- w_i 's word shape
- w_i 's short word shape
- w_i is upper case and has a digit and a dash (like *CFC-12*)
- w_i is upper case and followed within 3 words by Co., Inc., etc.

word shape

Word shape features are used to represent the abstract letter pattern of the word by mapping lower-case letters to 'x', upper-case to 'X', numbers to 'd', and retaining punctuation. Thus for example I.M.F would map to X.X.X. and DC10-30 would map to XXdd-dd. A second class of shorter word shape features is also used. In these features consecutive character types are removed, so DC10-30 would be mapped to Xd-d but I.M.F would still map to X.X.X. For example the word *well-dressed* would generate the following non-zero valued feature values:

```

prefix( $w_i$ ) = w
prefix( $w_i$ ) = we
prefix( $w_i$ ) = wel
prefix( $w_i$ ) = well
suffix( $w_i$ ) = ssed
suffix( $w_i$ ) = sed
suffix( $w_i$ ) = ed
suffix( $w_i$ ) = d
has-hyphen( $w_i$ )
word-shape( $w_i$ ) = xxxx-xxxxxxxx
short-word-shape( $w_i$ ) = x-x

```

Features for known words, like the templates in Eq. 8.33, are computed for every word seen in the training set. The unknown word features can also be computed for all words in training, or only on training words whose frequency is below some threshold. The result of the known-word templates and word-signature features is a very large set of features. Generally a feature cutoff is used in which features are thrown out if they have count < 5 in the training set.

8.5.2 Decoding and Training MEMMs

The most likely sequence of tags is then computed by combining these features of the input word w_i , its neighbors within l words w_{i-l}^{i+l} , and the previous k tags t_{i-k}^{i-1} as follows (using θ to refer to feature weights instead of w to avoid the confusion with w meaning words):

$$\begin{aligned}
\hat{T} &= \underset{T}{\operatorname{argmax}} P(T|W) \\
&= \underset{T}{\operatorname{argmax}} \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\
&= \underset{T}{\operatorname{argmax}} \prod_i \frac{\exp \left(\sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)}{\sum_{t' \in \text{tagset}} \exp \left(\sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1}) \right)} \quad (8.34)
\end{aligned}$$

How should we decode to find this optimal tag sequence \hat{T} ? The simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on. This is called a **greedy** decoding algorithm, because we greedily choose the best tag for each word, as shown in Fig. 8.14.

```

function GREEDY SEQUENCE DECODING(words W, model P) returns tag sequence T
for  $i = 1$  to  $\text{length}(W)$ 
     $\hat{t}_i = \underset{t' \in T}{\operatorname{argmax}} P(t' | w_{i-l}^{i+l}, t_{i-k}^{i-1})$ 

```

Figure 8.14 In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can't use evidence from future decisions. Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too great a drop in performance, and we don't use it.

Viterbi

Instead we decode an MEMM with the **Viterbi** algorithm just as with the HMM, finding the sequence of part-of-speech tags that is optimal for the whole sentence.

For example, assume that our MEMM is only conditioning on the previous tag t_{i-1} and observed word w_i . Concretely, this involves filling an $N \times T$ array with the appropriate values for $P(t_i|t_{i-1}, w_i)$, maintaining backpointers as we proceed. As with HMM Viterbi, when the table is filled, we simply follow pointers back from the maximum value in the final column to retrieve the desired set of labels. The requisite changes from the HMM-style application of Viterbi have to do only with how we fill each cell. Recall from Eq. 8.20 that the recursive step of the Viterbi equation computes the Viterbi value of time t for state j as

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \quad (8.35)$$

which is the HMM implementation of

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i) P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T \quad (8.36)$$

The MEMM requires only a slight change to this latter formula, replacing the a and b prior and likelihood probabilities with the direct posterior:

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) P(s_j|s_i, o_t) \quad 1 \leq j \leq N, 1 < t \leq T \quad (8.37)$$

Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression. Given a sequence of observations, feature functions, and corresponding hidden states, we use gradient descent to train the weights to maximize the log-likelihood of the training corpus.

8.6 Bidirectionality

The one problem with the MEMM and HMM models as presented is that they are exclusively run left-to-right. While the Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions, it would help even more if a decision about word w_i could directly use information about future tags t_{i+1} and t_{i+2} .

label bias
observation bias

Adding bidirectionality has another useful advantage. MEMMs have a theoretical weakness, referred to alternatively as the **label bias** or **observation bias** problem (Lafferty et al. 2001, Toutanova et al. 2003). These are names for situations when one source of information is ignored because it is **explained away** by another source. Consider an example from Toutanova et al. (2003), the sequence *will/NN to/TO fight/VB*. The tag TO is often preceded by NN but rarely by modals (MD), and so that tendency should help predict the correct NN tag for *will*. But the previous transition $P(t_{will}|\langle s \rangle)$ prefers the modal, and because $P(TO|to, t_{will})$ is so close to 1 regardless of t_{will} the model cannot make use of the transition probability and incorrectly chooses MD. The strong information that *to* must have the tag TO has **explained away** the presence of TO and so the model doesn't learn the importance of

the previous NN tag for predicting TO. Bidirectionality helps the model by making the link between TO available when tagging the NN.

CRF One way to implement bidirectionality is to switch to a more powerful model called a **conditional random field** or **CRF**. The CRF is an undirected graphical model, which means that it's not computing a probability for each tag at each time step. Instead, at each time step the CRF computes log-linear functions over a **clique**, a set of relevant features. Unlike for an MEMM, these might include output features of words in future time steps. The probability of the best sequence is similarly computed by the Viterbi algorithm. Because a CRF normalizes probabilities over all tag sequences, rather than over all the tags at an individual time t , training requires computing the sum over all possible labelings, which makes CRF training quite slow.

Stanford tagger Simpler methods can also be used; the **Stanford tagger** uses a bidirectional version of the MEMM called a cyclic dependency network (Toutanova et al., 2003).

Alternatively, any sequence model can be turned into a bidirectional model by using multiple passes. For example, the first pass would use only part-of-speech features from already-disambiguated words on the left. In the second pass, tags for all words, including those on the right, can be used. Alternately, the tagger can be run twice, once left-to-right and once right-to-left. In greedy decoding, for each word the classifier chooses the highest-scoring of the tags assigned by the left-to-right and right-to-left classifier. In Viterbi decoding, the classifier chooses the higher scoring of the two sequences (left-to-right or right-to-left). These bidirectional models lead directly into the bi-LSTM models that we will introduce in Chapter 9 as a standard neural sequence model.

8.7 Part-of-Speech Tagging for Morphological Rich Languages

Augmentations to tagging algorithms become necessary when dealing with languages with rich morphology like Czech, Hungarian and Turkish.

These productive word-formation processes result in a large vocabulary for these languages: a 250,000 word token corpus of Hungarian has more than twice as many word types as a similarly sized corpus of English (Oravecz and Dienes, 2002), while a 10 million word token corpus of Turkish contains four times as many word types as a similarly sized English corpus (Hakkani-Tür et al., 2002). Large vocabularies mean many unknown words, and these unknown words cause significant performance degradations in a wide variety of languages (including Czech, Slovene, Estonian, and Romanian) (Hajič, 2000).

Highly inflectional languages also have much more information than English coded in word morphology, like **case** (nominative, accusative, genitive) or **gender** (masculine, feminine). Because this information is important for tasks like parsing and coreference resolution, part-of-speech taggers for morphologically rich languages need to label words with case and gender information. Tagsets for morphologically rich languages are therefore sequences of morphological tags rather than a single primitive tag. Here's a Turkish example, in which the word *izin* has three possible morphological/part-of-speech tags and meanings (Hakkani-Tür et al., 2002):

- | | |
|--|--|
| <ol style="list-style-type: none"> 1. Yerdeki izin temizlenmesi gereklidir.
The trace on the floor should be cleaned. 2. Üzerinde parmak izin kalmış
iz + Noun+A3sg+Pnon+Gen | <ol style="list-style-type: none"> 1. Yerdeki izin temizlenmesi gereklidir.
The trace on the floor should be cleaned. 2. Üzerinde parmak izin kalmış
iz + Noun+A3sg+P2sg+Nom |
|--|--|

Your finger **print** is left on (it).

3. İçeri girmek için **izin** alman gerekiyor. izin + Noun+A3sg+Pnon+Nom
You need **permission** to enter.

Using a morphological parse sequence like Noun+A3sg+Pnon+Gen as the part-of-speech tag greatly increases the number of parts of speech, and so tagsets can be 4 to 10 times larger than the 50–100 tags we have seen for English. With such large tagsets, each word needs to be morphologically analyzed to generate the list of possible morphological tag sequences (part-of-speech tags) for the word. The role of the tagger is then to disambiguate among these tags. This method also helps with unknown words since morphological parsers can accept unknown stems and still segment the affixes properly.

For non-word-space languages like Chinese, word segmentation (Chapter 2) is either applied before tagging or done jointly. Although Chinese words are on average very short (around 2.4 characters per unknown word compared with 7.7 for English) the problem of unknown words is still large. While English unknown words tend to be proper nouns in Chinese the majority of unknown words are common nouns and verbs because of extensive compounding. Tagging models for Chinese use similar unknown word features to English, including character prefix and suffix features, as well as novel features like the **radicals** of each character in a word. (Tseng et al., 2005).

A standard for multilingual tagging is the Universal POS tag set of the Universal Dependencies project, which contains 16 tags plus a wide variety of features that can be added to them to create a large tagset for any language (Nivre et al., 2016a).

8.8 Summary

This chapter introduced **parts of speech** and **part-of-speech tagging**:

- Languages generally have a small set of **closed class** words that are highly frequent, ambiguous, and act as **function words**, and **open-class** words like **nouns**, **verbs**, **adjectives**. Various part-of-speech **tagsets** exist, of between 40 and 200 tags.
- **Part-of-speech tagging** is the process of assigning a part-of-speech label to each of a sequence of words.
- Two common approaches to **sequence modeling** are a **generative** approach, **HMM** tagging, and a **discriminative** approach, **MEMM** tagging. We will see a third, discriminative neural approach in Chapter 9.
- The probabilities in HMM taggers are estimated by maximum likelihood estimation on tag-labeled training corpora. The Viterbi algorithm is used for **decoding**, finding the most likely tag sequence
- Beam search is a variant of Viterbi decoding that maintains only a fraction of high scoring states rather than all states during decoding.
- **Maximum entropy Markov model** or **MEMM taggers** train logistic regression models to pick the best tag given an observation word and its context and the previous tags, and then use Viterbi to choose the best sequence of tags.
- Modern taggers are generally run **bidirectionally**.

Bibliographical and Historical Notes

What is probably the earliest part-of-speech tagger was part of the parser in Zellig Harris's Transformations and Discourse Analysis Project (TDAP), implemented between June 1958 and July 1959 at the University of Pennsylvania (Harris, 1962), although earlier systems had used part-of-speech dictionaries. TDAP used 14 handwritten rules for part-of-speech disambiguation; the use of part-of-speech tag sequences and the relative frequency of tags for a word prefigures all modern algorithms. The parser was implemented essentially as a cascade of finite-state transducers; see Joshi and Hopely (1999) and Karttunen (1999) for a reimplementation.

The Computational Grammar Coder (CGC) of Klein and Simmons (1963) had three components: a lexicon, a morphological analyzer, and a context disambiguator. The small 1500-word lexicon listed only function words and other irregular words. The morphological analyzer used inflectional and derivational suffixes to assign part-of-speech classes. These were run over words to produce candidate parts of speech which were then disambiguated by a set of 500 context rules by relying on surrounding islands of unambiguous words. For example, one rule said that between an ARTICLE and a VERB, the only allowable sequences were ADJ-NOUN, NOUN-ADVERB, or NOUN-NOUN. The TAGGIT tagger (Greene and Rubin, 1971) used the same architecture as Klein and Simmons (1963), with a bigger dictionary and more tags (87). TAGGIT was applied to the Brown corpus and, according to Francis and Kučera (1982, p. 9), accurately tagged 77% of the corpus; the remainder of the Brown corpus was then tagged by hand. All these early algorithms were based on a two-stage architecture in which a dictionary was first used to assign each word a set of potential parts of speech, and then lists of handwritten disambiguation rules winnowed the set down to a single part of speech per word.

Soon afterwards probabilistic architectures began to be developed. Probabilities were used in tagging by Stoltz et al. (1965) and a complete probabilistic tagger with Viterbi decoding was sketched by Bahl and Mercer (1976). The Lancaster-Oslo/Bergen (LOB) corpus, a British English equivalent of the Brown corpus, was tagged in the early 1980's with the CLAWS tagger (Marshall 1983; Marshall 1987; Garside 1987), a probabilistic algorithm that approximated a simplified HMM tagger. The algorithm used tag bigram probabilities, but instead of storing the word likelihood of each tag, the algorithm marked tags either as *rare* ($P(\text{tag}|\text{word}) < .01$) *infrequent* ($P(\text{tag}|\text{word}) < .10$) or *normally frequent* ($P(\text{tag}|\text{word}) > .10$).

DeRose (1988) developed a quasi-HMM algorithm, including the use of dynamic programming, although computing $P(t|w)P(w)$ instead of $P(w|t)P(w)$. The same year, the probabilistic PARTS tagger of Church (1988), (1989) was probably the first implemented HMM tagger, described correctly in Church (1989), although Church (1988) also described the computation incorrectly as $P(t|w)P(w)$ instead of $P(w|t)P(w)$. Church (p.c.) explained that he had simplified for pedagogical purposes because using the probability $P(t|w)$ made the idea seem more understandable as "storing a lexicon in an almost standard form".

Later taggers explicitly introduced the use of the hidden Markov model (Kupiec 1992; Weischedel et al. 1993; Schütze and Singer 1994). Merialdo (1994) showed that fully unsupervised EM didn't work well for the tagging task and that reliance on hand-labeled data was important. Charniak et al. (1993) showed the importance of the most frequent tag baseline; the 92.3% number we give above was from Abney et al. (1999). See Brants (2000) for many implementation details of an HMM tagger whose performance is still roughly close to state of the art taggers.

Ratnaparkhi (1996) introduced the MEMM tagger, called MXPOST, and the modern formulation is very much based on his work.

The idea of using letter suffixes for unknown words is quite old; the early Klein and Simmons (1963) system checked all final letter suffixes of lengths 1-5. The probabilistic formulation we described for HMMs comes from Samuelsson (1993). The unknown word features described on page 161 come mainly from (Ratnaparkhi, 1996), with augmentations from Toutanova et al. (2003) and Manning (2011).

State of the art taggers use neural algorithms like the sequence models in Chapter 9 or (bidirectional) log-linear models Toutanova et al. (2003). HMM (Brants 2000; Thede and Harper 1999) and MEMM tagger accuracies are likely just a tad lower.

An alternative modern formalism, the English Constraint Grammar systems (Karls-son et al. 1995; Voutilainen 1995; Voutilainen 1999), uses a two-stage formalism much like the early taggers from the 1950s and 1960s. A morphological analyzer with tens of thousands of English word stem entries returns all parts of speech for a word, using a large feature-based tagset. So the word *occurred* is tagged with the options ⟨V PCP2 SV⟩ and ⟨V PAST VFIN SV⟩, meaning it can be a participle (PCP2) for an intransitive (SV) verb, or a past (PAST) finite (VFIN) form of an intransitive (SV) verb. A set of 3,744 constraints are then applied to the input sentence to rule out parts of speech inconsistent with the context. For example here's a rule for the ambiguous word *that* that eliminates all tags except the ADV (adverbial intensifier) sense (this is the sense in the sentence *it isn't that odd*):

```
ADVERBIAL-THE RULE Given input: "that"
if (+1 A/ADV/QUANT); /* if next word is adj, adverb, or quantifier */
/* (+2 SENT-LIM); /* and following which is a sentence boundary, */
/* (NOT -1 SVOC/A); /* and the previous word is not a verb like */
/* /* 'consider' which allows adjs as object complements */
then eliminate non-ADV tags else eliminate ADV tag
```

Manning (2011) investigates the remaining 2.7% of errors in a high-performing tagger, the bidirectional MEMM-style model described above (Toutanova et al., 2003). He suggests that a third or half of these remaining errors are due to errors or inconsistencies in the training data, a third might be solvable with richer linguistic models, and for the remainder the task is underspecified or unclear.

Supervised tagging relies heavily on in-domain training data hand-labeled by experts. Ways to relax this assumption include unsupervised algorithms for clustering words into part-of-speech-like classes, summarized in Christodoulopoulos et al. (2010), and ways to combine labeled and unlabeled data, for example by co-training (Clark et al. 2003; Søgaard 2010).

See Householder (1995) for historical notes on parts of speech, and Sampson (1987) and Garside et al. (1997) on the provenance of the Brown and other tagsets.

Exercises

8.1 Find one tagging error in each of the following sentences that are tagged with the Penn Treebank tagset:

1. I/PRP need/VBP a/DT flight/NN from/IN Atlanta/NN
2. Does/VBZ this/DT flight/NN serve/VB dinner/NNS
3. I/PRP have/VB a/DT friend/NN living/VBG in/IN Denver/NNP
4. Can/VBP you/PRP list/VB the/DT nonstop/JJ afternoon/NN flights/NNS

- 8.2** Use the Penn Treebank tagset to tag each word in the following sentences from Damon Runyon’s short stories. You may ignore punctuation. Some of these are quite difficult; do your best.
1. It is a nice night.
 2. This crap game is over a garage in Fifty-second Street...
 3. ...Nobody ever takes the newspapers she sells ...
 4. He is a tall, skinny guy with a long, sad, mean-looking kisser, and a mournful voice.
 5. ...I am sitting in Mindy’s restaurant putting on the gefillte fish, which is a dish I am very fond of, ...
 6. When a guy and a doll get to taking peeks back and forth at each other, why there you are indeed.
- 8.3** Now compare your tags from the previous exercise with one or two friend’s answers. On which words did you disagree the most? Why?
- 8.4** Implement the “most likely tag” baseline. Find a POS-tagged training set, and use it to compute for each word the tag that maximizes $p(t|w)$. You will need to implement a simple tokenizer to deal with sentence boundaries. Start by assuming that all unknown words are NN and compute your error rate on known and unknown words. Now write at least five rules to do a better job of tagging unknown words, and show the difference in error rates.
- 8.5** Build a bigram HMM tagger. You will need a part-of-speech-tagged corpus. First split the corpus into a training set and test set. From the labeled training set, train the transition and observation probabilities of the HMM tagger directly on the hand-tagged data. Then implement the Viterbi algorithm so that you can label an arbitrary test sentence. Now run your algorithm on the test set. Report its error rate and compare its performance to the most frequent tag baseline.
- 8.6** Do an error analysis of your tagger. Build a confusion matrix and investigate the most frequent errors. Propose some features for improving the performance of your tagger on these errors.

Sequence Processing with Recurrent Networks

Time will explain.
Jane Austen, *Persuasion*

Language is an inherently temporal phenomenon. When we comprehend and produce spoken language, we are processing continuous input streams of indefinite length. And even when dealing with written text we normally process it sequentially, even though we in principle have arbitrary access to all the elements at once. The temporal nature of language is reflected in the metaphors we use; we talk of the *flow of conversations*, *news feeds*, and *twitter streams*, all of which call out the notion that language is a sequence that unfolds in time. This temporal nature is also reflected in the algorithms we use to process language. When applied to the problem of part-of-speech tagging, the Viterbi algorithm works its way incrementally through its input a word at a time, taking into account information gleaned along the way. The syntactic parsing algorithms we cover in Chapters 11, 12, and 13 operate in a similar fashion.

In contrast, the machine learning approaches we've studied for sentiment analysis and other classification tasks do not have this temporal nature. These approaches have simultaneous access to all aspects of their input. This is certainly true of feed-forward neural networks, including their application to neural language models. Such networks employ fixed-size input vectors with associated weights to capture all relevant aspects of an example at once. This makes it difficult to deal with sequences of varying length, and they fail to capture important temporal aspects of language.

We saw one work-around for these problems with the case of neural language models. These models operate by accepting fixed-sized windows of tokens as input; sequences longer than the window size are processed by sliding windows over the input making predictions as they go, with the end result being a sequence of predictions spanning the input. Importantly, the decision made for one window has no impact on later decisions. Fig. 9.1, reproduced here from Chapter 7, illustrates this approach with a window of size 3. Here, we're predicting which word will come next given the window *the ground there*. Subsequent words are predicted by sliding the window forward one word at a time.

The sliding window approach is problematic for a number of reasons. First, it shares the primary weakness of Markov approaches in that it limits the context from which information can be extracted; anything outside the context window has no impact on the decision being made. This is an issue since there are many language tasks that require access to information that can be arbitrarily distant from the point at which processing is happening. Second, the use of windows makes it difficult for networks to learn systematic patterns arising from phenomena like constituency. For

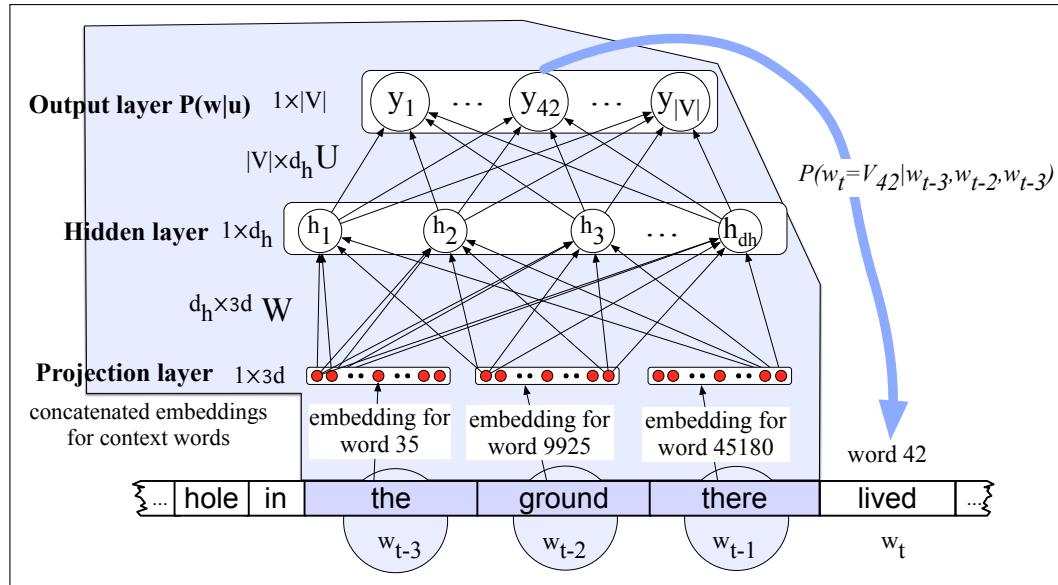


Figure 9.1 A simplified view of a feedforward neural language model moving through a text. At each time step t the network takes the 3 context words, converts each to a d -dimensional embedding, and concatenates the 3 embeddings together to get the $1 \times Nd$ unit input layer x for the network.

example, in Fig. 9.1 the noun phrase *the ground* appears in two separate windows: once, as shown, in the first and second positions in the window, and in the preceding step where it appears in the second and third positions, thus forcing the network to learn two separate patterns for what should be a constituent.

The subject of this chapter is **recurrent neural networks**, a class of networks designed to address these challenges by dealing directly with the temporal aspect of language, allowing us to handle variable length inputs without the use of arbitrary fixed-sized windows, and providing the means to capture and exploit the temporal nature of language.

9.1 Simple Recurrent Neural Networks

Elman
Networks

A recurrent neural network (RNN) is any network that contains a cycle within its network connections. That is, any network where the value of a unit is directly, or indirectly, dependent on earlier outputs as an input. While powerful, such networks are difficult to reason about and to train. However, within the general class of recurrent networks there are constrained architectures that have proven to be extremely effective when applied to spoken and written language. In this section, we consider a class of recurrent networks referred to as **Elman Networks** (Elman, 1990) or **simple recurrent networks**. These networks are useful in their own right and serve as the basis for more complex approaches to be discussed later in this chapter and again in Chapter 10 and Chapter 11. Going forward, when we use the term RNN we'll be referring to these simpler more constrained networks.

Fig. 9.2 illustrates the structure of a simple RNN. As with ordinary feedforward networks, an input vector representing the current input element, x_t , is multiplied by a weight matrix and then passed through an activation function to compute an activa-

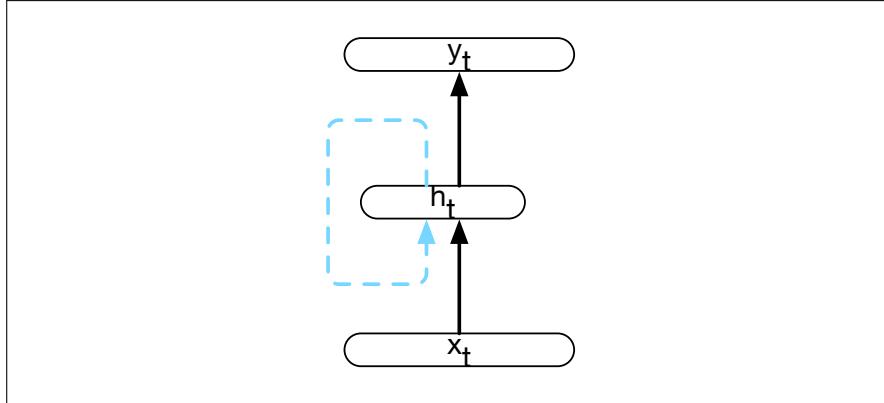


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous time step.

tion value for a layer of hidden units. This hidden layer is, in turn, used to calculate a corresponding output, y_t . In a departure from our earlier window-based approach, sequences are processed by presenting one element at a time to the network. The key difference from a feedforward network lies in the recurrent link shown in the figure with the dashed line. This link augments the input to the computation at the hidden layer with the activation value of the hidden layer *from the preceding point in time*.

The hidden layer from the previous time step provides a form of memory, or context, that encodes earlier processing and informs the decisions to be made at later points in time. Critically, this architecture does not impose a fixed-length limit on this prior context; the context embodied in the previous hidden layer includes information extending back to the beginning of the sequence.

Adding this temporal dimension may make RNNs appear to be more exotic than non-recurrent architectures. But in reality, they’re not all that different. Given an input vector and the values for the hidden layer from the previous time step, we’re still performing the standard feedforward calculation. To see this, consider Fig. 9.3 which clarifies the nature of the recurrence and how it factors into the computation at the hidden layer. The most significant change lies in the new set of weights, U , that connect the hidden layer from the previous time step to the current hidden layer. These weights determine how the network should make use of past context in calculating the output for the current input. As with the other weights in the network, these connections are trained via backpropagation.

9.1.1 Inference in Simple RNNs

Forward inference (mapping a sequence of inputs to a sequence of outputs) in an RNN is nearly identical to what we’ve already seen with feedforward networks. To compute an output y_t for an input x_t , we need the activation value for the hidden layer h_t . To calculate this, we multiply the input x_t with the weight matrix W , and the hidden layer from the previous time step h_{t-1} with the weight matrix U . We add these values together and pass them through a suitable activation function, g , to arrive at the activation value for the current hidden layer, h_t . Once we have the values for the hidden layer, we proceed with the usual computation to generate the

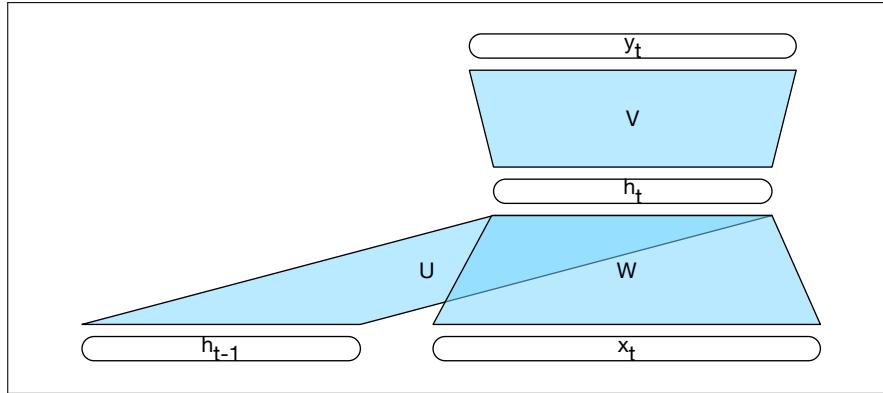


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

output vector.

$$\begin{aligned} h_t &= g(Uh_{t-1} + Wh_t) \\ y_t &= f(Vh_t) \end{aligned}$$

In the commonly encountered case of soft classification, computing y_t consists of a softmax computation that provides a normalized probability distribution over the possible output classes.

$$y_t = \text{softmax}(Vh_t)$$

The fact that the computation at time t requires the value of the hidden layer from time $t - 1$ mandates an incremental inference algorithm that proceeds from the start of the sequence to the end as illustrated in Fig. 9.4. The sequential nature of simple recurrent networks can also be seen by *unrolling* the network in time as is shown in Fig. 9.5. In this figure, the various layers of units are copied for each time step to illustrate that they will have differing values over time. However, the various weight matrices are shared across time.

```

function FORWARDRNN( $x, network$ ) returns output sequence  $y$ 
     $h_0 \leftarrow 0$ 
    for  $i \leftarrow 1$  to LENGTH( $x$ ) do
         $h_i \leftarrow g(Uh_{i-1} + Wh_i)$ 
         $y_i \leftarrow f(Vh_i)$ 
    return  $y$ 

```

Figure 9.4 Forward inference in a simple recurrent network. The matrices U , V and W are shared across time, while new values for h and y are calculated with each time step.

9.1.2 Training

As with feedforward networks, we'll use a training set, a loss function, and backpropagation to obtain the gradients needed to adjust the weights in these recurrent networks. As shown in Fig. 9.3, we now have 3 sets of weights to update: W , the

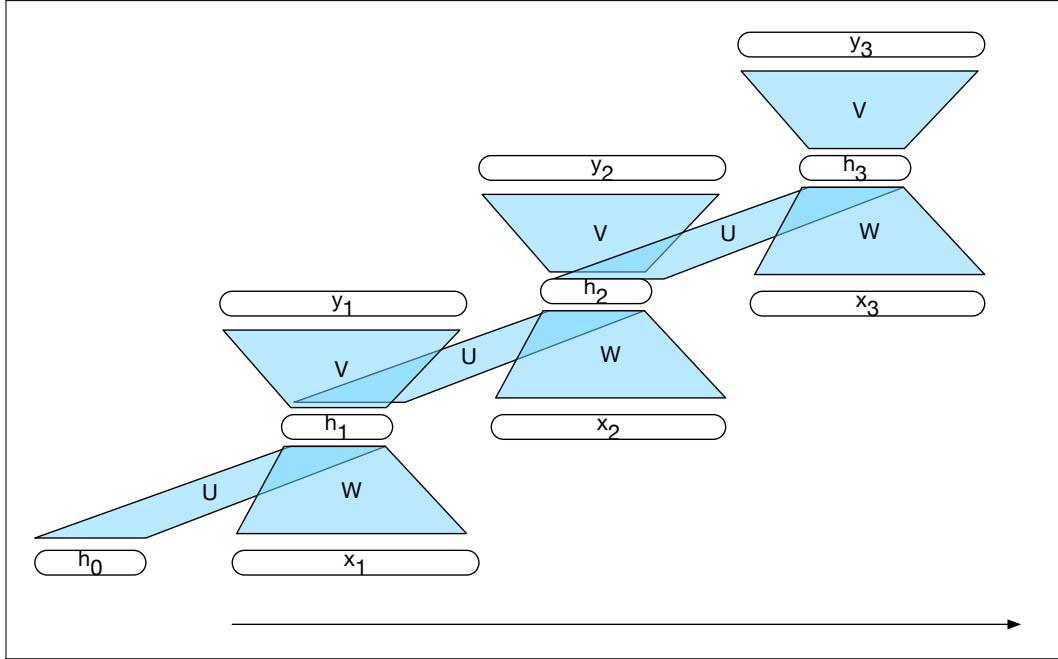


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U , V and W are shared in common across all time steps.

weights from the input layer to the hidden layer, U , the weights from the previous hidden layer to the current hidden layer, and finally V , the weights from the hidden layer to the output layer.

Before going on, let's first review some of the notation that we introduced in Chapter 7. Assuming a network with an input layer x and a non-linear activation function g , $a^{[i]}$ refers to the activation value from a layer i , which is the result of applying g to $z^{[i]}$, the weighted sum of the inputs to that layer.

Fig. 9.5 illustrates two considerations that we didn't have to worry about with backpropagation in feedforward networks. First, to compute the loss function for the output at time t we need the hidden layer from time $t - 1$. Second, the hidden layer at time t influences both the output at time t and the hidden layer at time $t + 1$ (and hence the output and loss at $t + 1$). It follows from this that to assess the error accruing to h_t , we'll need to know its influence on both the current output *as well as the ones that follow*.

Consider the situation where we are examining an input/output pair at time 2 as shown in Fig. 9.6. What do we need to compute the gradients required to update the weights U , V , and W here? Let's start by reviewing how we compute the gradients required to update V since this computation is unchanged from feedforward networks. To review from Chapter 7, we need to compute the derivative of the loss function L with respect to the weights V . However, since the loss is not expressed directly in terms of the weights, we apply the chain rule to get there indirectly.

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial V}$$

The first term on the right is the derivative of the loss function with respect to the network output, a . The second term is the derivative of the network output with respect to the intermediate network activation z , which is a function of the activation

function g . The final term in our application of the chain rule is the derivative of the network activation with respect to the weights V , which is the activation value of the current hidden layer h_t .

It's useful here to use the first two terms to define δ , an error term that represents how much of the scalar loss is attributable to each of the units in the output layer.

$$\delta_{out} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z} \quad (9.1)$$

$$\delta_{out} = L' g'(z) \quad (9.2)$$

Therefore, the final gradient we need to update the weight matrix V is just:

$$\frac{\partial L}{\partial V} = \delta_{out} h_t \quad (9.3)$$

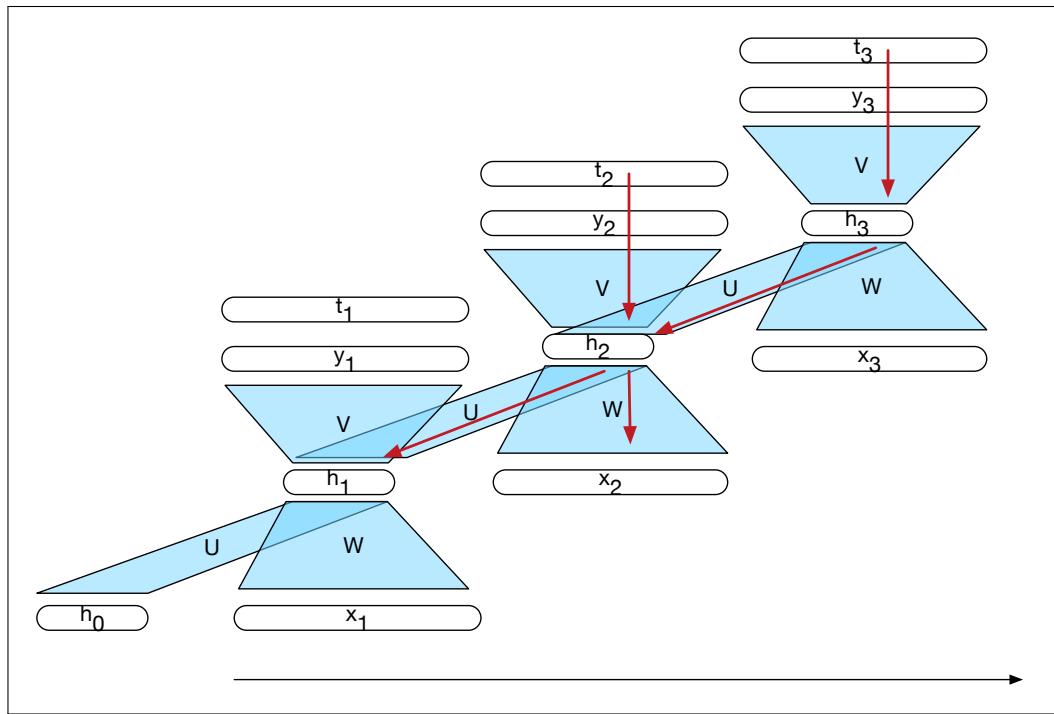


Figure 9.6 The backpropagation of errors in a simple RNN. t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U , V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

Moving on, we need to compute the corresponding gradients for the weight matrices W and U : $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial U}$. Here we encounter the first substantive change from feedforward networks. The hidden state at time t contributes to the output and associated error at time t and to the output and error at the next time step, $t + 1$. Therefore, the error term, δ_h , for the hidden layer must be the sum of the error term from the current output and its error from the next time step.

$$\delta_h = g'(z)V\delta_{out} + \delta_{next}$$

Given this total error term for the hidden layer, we can compute the gradients for the weights U and W using the chain rule as we did in Chapter 7.

$$\begin{aligned}\frac{dL}{dW} &= \frac{dL}{dz} \frac{dz}{da} \frac{da}{dW} \\ \frac{dL}{dU} &= \frac{dL}{dz} \frac{dz}{da} \frac{da}{dU}\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial W} &= \delta_h x_t \\ \frac{\partial L}{\partial U} &= \delta_h h_{t-1}\end{aligned}$$

These gradients provide us with the information needed to update the matrices U and W .

We're not quite done yet, we still need to assign proportional blame (compute the error term) back to the previous hidden layer h_{t-1} for use in further processing. This involves backpropagating the error from δ_h to h_{t-1} proportionally based on the weights in U .

$$\delta_{next} = g'(z)U\delta_h \quad (9.4)$$

At this point we have all the gradients needed to perform weight updates for each of our three sets of weights. Note that in this simple case there is no need to backpropagate the error through W to the input x , since the input training data is assumed to be fixed. If we wished to update our input word or character embeddings we would backpropagate the error through to them as well.

Taken together, all of these considerations lead to a two-pass algorithm for training the weights in RNNs. In the first pass, we perform forward inference, computing h_t , y_t , and accumulating the loss at each step in time, saving the value of the hidden layer at each step for use at the next time step. In the second phase, we process the sequence in reverse, computing the required error terms gradients as we go, computing and saving the error term for use in the hidden layer for each step backward in time. This general approach is commonly referred to as **Backpropagation Through Time** (Werbos 1974, Rumelhart et al. 1986, Werbos 1990).

Backpropagation Through Time

9.1.3 Unrolled Networks as Computation Graphs

We used the unrolled network shown in Fig. 9.5 as a way to illustrate the temporal nature of RNNs. However, with modern computational frameworks and adequate computing resources, explicitly unrolling a recurrent network into a deep feedforward computational graph is quite practical for word-by-word approaches to sentence-level processing. In such an approach, we provide a template that specifies the basic structure of the network, including all the necessary parameters for the input, output, and hidden layers, the weight matrices, as well as the activation and output functions to be used. Then, when presented with a particular input sequence, we can generate an unrolled feedforward network specific to that input, and use that graph to perform forward inference or training via ordinary backpropagation.

For applications that involve much longer input sequences, such as speech recognition, character-by-character sentence processing, or streaming of continuous inputs, unrolling an entire input sequence may not be feasible. In these cases, we can unroll the input into manageable fixed-length segments and treat each segment as a distinct training item.

9.2 Applications of Recurrent Neural Networks

Recurrent neural networks have proven to be an effective approach to language modeling, sequence labeling tasks such as part-of-speech tagging, as well as sequence classification tasks such as sentiment analysis and topic classification. And as we'll see in Chapter 10 and Chapter 11, they form the basis for sequence-to-sequence approaches to summarization, machine translation, and question answering.

9.2.1 Recurrent Neural Language Models

We've already seen two ways to create probabilistic language models: N -gram models and feedforward networks with sliding windows. Given a fixed preceding context, both attempt to predict the next word in a sequence. More formally, they compute the conditional probability of the next word in a sequence given the preceding words, $P(w_n|w_1^{n-1})$.

In both approaches, the quality of a model is largely dependent on the size of the context and how effectively the model makes use of it. Thus, both N -gram and sliding-window neural networks are constrained by the Markov assumption embodied in the following equation.

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \quad (9.5)$$

That is, anything outside the preceding context of size N has no bearing on the computation.

Recurrent neural language models process sequences a word at a time attempting to predict the next word in a sequence by using the current word and the previous hidden state as input (Mikolov et al., 2010). Thus, the limited context constraint inherent in both N -gram models and sliding window approaches is avoided since the hidden state embodies information about all of the preceding words all the way back to the beginning of the sequence.

Forward inference in a recurrent language model proceeds as described in Section 9.1.1. At each step the network retrieves a word embedding for the current word as input and combines it with the hidden layer from the previous step to compute a new hidden layer. This hidden layer is then used to generate an output layer which is passed through a softmax layer to generate a probability distribution over the entire vocabulary.

$$P(w_n|w_1^{n-1}) = y_n \quad (9.6)$$

$$= \text{softmax}(Vh_n) \quad (9.7)$$

The probability of an entire sequence is then just the product of the probabilities of each item in the sequence.

$$P(w_1^n) = \prod_{k=1}^n P(w_k|w_1^{k-1}) \quad (9.8)$$

$$= \prod_{k=1}^n y_k \quad (9.9)$$

As with the approach introduced in Chapter 7, to train such a model we use a corpus of representative text as training material. The task is to predict the next word in a sequence given the previous words, using cross-entropy as the loss function. Recall that the cross-entropy loss for a single example is the negative log probability assigned to the correct class, which is the result of applying a softmax to the final output layer.

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i \quad (9.10)$$

$$= -\log \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (9.11)$$

Here, the correct class i is the word that actually comes next in the data and y_i is the probability assigned to that word, and the softmax is over the entire vocabulary, which has size K . The weights in the network are adjusted to minimize the cross-entropy loss over the training set via gradient descent.

Generation with Neural Language Models

As we saw with the probabilistic Shakespeare generator in Chapter 3, a fun way to gain insight into a language model is to use Shannon’s method (Shannon, 1951) to generate random sentences. The procedure is basically the same as that described on 39.

- To begin, sample the first word in the output from the softmax distribution that results from using the beginning of sentence marker, $\langle s \rangle$, as the first input.
- Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.
- Continue generating until the end of sentence marker, $\langle /s \rangle$, is sampled or a fixed length limit is reached.

autoregressive generation

This technique is called **autoregressive generation** since the word generated at the each time step is conditioned on the word generated by the network at the previous step. Fig. 9.7 illustrates this approach. In this figure, the details of the RNN’s hidden layers and recurrent connections are hidden within the blue block.

While this is an entertaining exercise, this architecture has inspired state-of-the-art approaches to applications such as machine translation, summarization, and question answering. The key to these approaches is to prime the generation component with an appropriate context. That is, instead of simply using $\langle s \rangle$ to get things started we can provide a richer task-appropriate context. We’ll return to these more advanced applications in Chapter 10, where we discuss encoder-decoder networks.

Finally, as we did with Shakespeare, we can move beyond informally assessing the quality of generated output by using perplexity to objectively compare the output to a held-out sample of the training corpus.

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1})}} \quad (9.12)$$

The lower the perplexity, the better the model.

9.2.2 Sequence Labeling

In sequence labeling, the network’s task is to assign a label chosen from a small fixed set of labels to each element of a sequence. The canonical example of such a

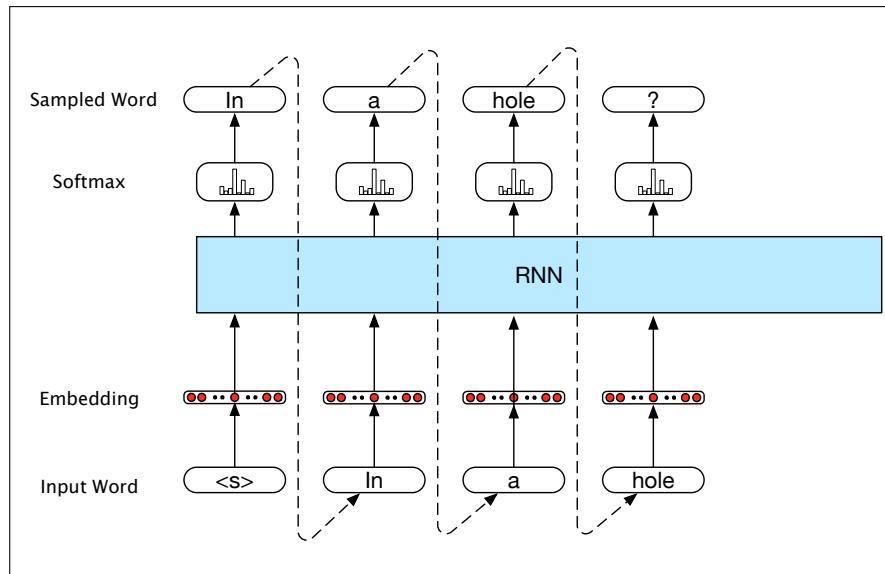


Figure 9.7 Autoregressive generation with an RNN-based neural language model.

task is part-of-speech tagging, discussed in detail in Chapter 8. In an RNN approach to POS tagging, inputs are word embeddings and the outputs are tag probabilities generated by a softmax layer over the tagset, as illustrated in Fig. 9.8.

In this figure, the inputs at each time step are pre-trained word embeddings corresponding to the input tokens. The RNN block is an abstraction that represents an unrolled simple recurrent network consisting of an input layer, hidden layer, and output layer at each time step, as well as the shared U , V and W weight matrices that comprise the network. The outputs of the network at each time step represent the distribution over the POS tagset generated by a softmax layer.

To generate a tag sequence for a given input, we can run forward inference over

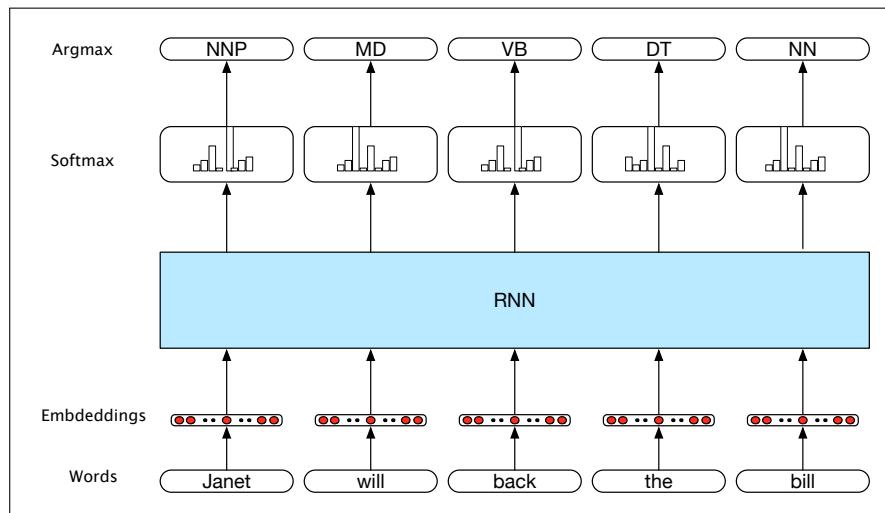


Figure 9.8 Part-of-speech tagging as sequence labeling with a simple RNN. Pre-trained word embeddings serve as inputs and a softmax layer provides a probability distribution over the part-of-speech tags as output at each time step.

the input sequence and select the most likely tag from the softmax at each step. Since we're using a softmax layer to generate the probability distribution over the output tagset at each time step, we will again employ the cross-entropy loss during training.

named entity recognition

A closely related, and extremely useful, application of sequence labeling is to find and classify *spans* of text that correspond to concepts of interest in some task domain. One example of such a task is **named entity recognition** — the problem of finding all the spans in a text that correspond to names of people, places or organizations (a problem we'll study in more detail in Chapter 18).

To use sequence labeling for a span-recognition problem, we'll use a technique called IOB encoding ([Ramshaw and Marcus, 1995](#)). In its simplest form, we label any token that *begins* a span of interest with the label B, tokens that occur *inside* a span are tagged with an I, and any tokens outside of any span of interest are labeled O. Consider the following example:

(9.13) *United cancelled the flight from Denver to San Francisco.*

B O O O O B O B I

Here, the spans of interest are *United*, *Denver* and *San Francisco*.

In applications where we are interested in more than one class of entity (e.g., finding and distinguishing names of people, locations, or organizations), we can specialize the B and I tags to represent each of the more specific classes, thus expanding the tagset from 3 tags to $2 * N + 1$, where N is the number of classes we're interested in. Applying this approach to the previous example results in the following encoding.

(9.14) *United cancelled the flight from Denver to San Francisco.*

B-ORG O O O B-LOC O B-LOC I-LOC

Given such an encoding, we've reduced the span recognition task to a per-word labeling task where the inputs are our usual word embeddings and the output consists of a sequence of softmax distributions over the tags at each point in the sequence.

structure prediction

Yet another application of sequence labeling is to the problem of **structure prediction**. Here the task is to take an input sequence and produce some kind of structured output, such as a parse tree or meaning representation. One way to model problems like this is to learn a sequence of actions, or operators, which when executed would produce the desired structure. Therefore, instead of predicting a label for each element of an input sequence, the network is trained to select a sequence of actions, which when executed in sequence produce the desired output. The clearest example of this approach is transition-based parsing which borrows the shift-reduce paradigm from compiler construction. We'll return to this application in Chapter 15 when we take up dependency parsing.

Viterbi and Conditional Random Fields (CRFs)

As we saw when we applied logistic regression to part-of-speech tagging, choosing the maximum probability label for each element in a sequence independently does not necessarily result in an optimal (or even very good) sequence of tags. In the case of IOB tagging, it doesn't even guarantee that the resulting sequence will be well-formed. For example, nothing in approach described in the last section prevents an output sequence from containing an I following an O, even though such a transition is illegal. Similarly, when dealing with multiple classes nothing would prevent an I-LOC tag from following a B-PER tag.

One solution to this problem is to combine the sequence of outputs from a recurrent network with an output-level language model as discussed in Chapter 8. We can then use a variant of the Viterbi algorithm to select the most likely tag sequence. This approach is usually implemented by adding a CRF (Lample et al., 2016a) layer as the final layer of recurrent network.

9.2.3 RNNs for Sequence Classification

Another use of RNNs is to classify entire sequences rather than the tokens within them. We've already encountered this task in Chapter 4 with our discussion of sentiment analysis. Other examples include document-level topic classification, spam detection, message routing for customer service applications, and deception detection. In all of these applications, sequences of text are classified as belonging to one of a small number of categories.

To apply RNNs in this setting, the text to be classified is passed through the RNN a word at a time generating a new hidden layer at each time step. The hidden layer for the final element of the text, h_n , is taken to constitute a compressed representation of the entire sequence. In the simplest approach to classification, h_n , serves as the input to a subsequent feedforward network that chooses a class via a softmax over the possible classes. Fig. 9.9 illustrates this approach.

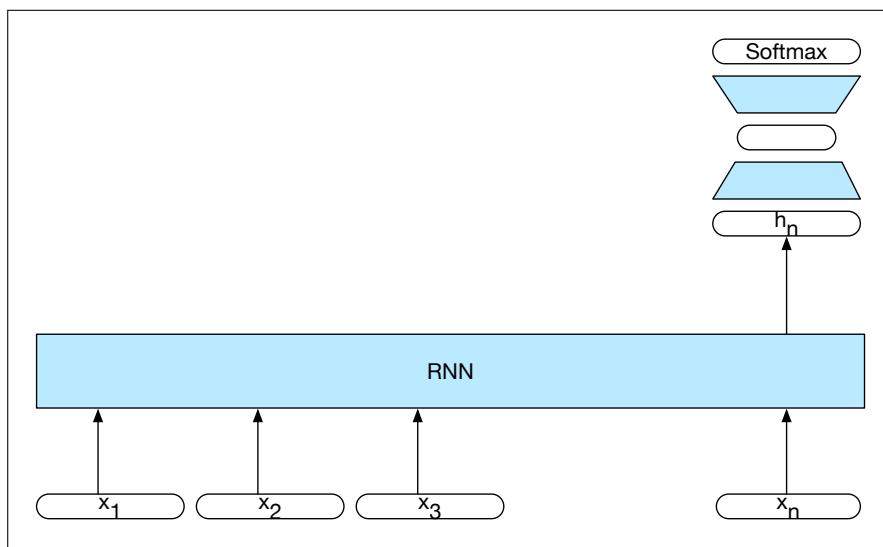


Figure 9.9 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Note that in this approach there are no intermediate outputs for the words in the sequence preceding the last element. Therefore, there are no loss terms associated with those elements. Instead, the loss function used to train the weights in the network is based entirely on the final text classification task. Specifically, the output from the softmax output from the feedforward classifier together with a cross-entropy loss drives the training. The error signal from the classification is backpropagated all the way through the weights in the feedforward classifier through, to its input, and then through to the three sets of weights in the RNN as described earlier in Section 9.1.2. This combination of a simple recurrent network with a feedforward

end-to-end
training

classifier is our first example of a *deep neural network*. And the training regimen that uses the loss from a downstream application to adjust the weights all the way through the network is referred to as **end-to-end training**.

9.3 Deep Networks: Stacked and Bidirectional RNNs

As suggested by the sequence classification architecture shown in Fig. 9.9, recurrent networks are quite flexible. By combining the feedforward nature of unrolled computational graphs with vectors as common inputs and outputs, complex networks can be treated as modules that can be combined in creative ways. This section introduces two of the more common network architectures used in language processing with RNNs.

9.3.1 Stacked RNNs

Stacked RNNs

In our examples thus far, the inputs to our RNNs have consisted of sequences of word or character embeddings (vectors) and the outputs have been vectors useful for predicting words, tags or sequence labels. However, nothing prevents us from using the entire sequence of outputs from one RNN as an input sequence to another one. **Stacked RNNs** consist of multiple networks where the output of one layer serves as the input to a subsequent layer, as shown in Fig. 9.10.

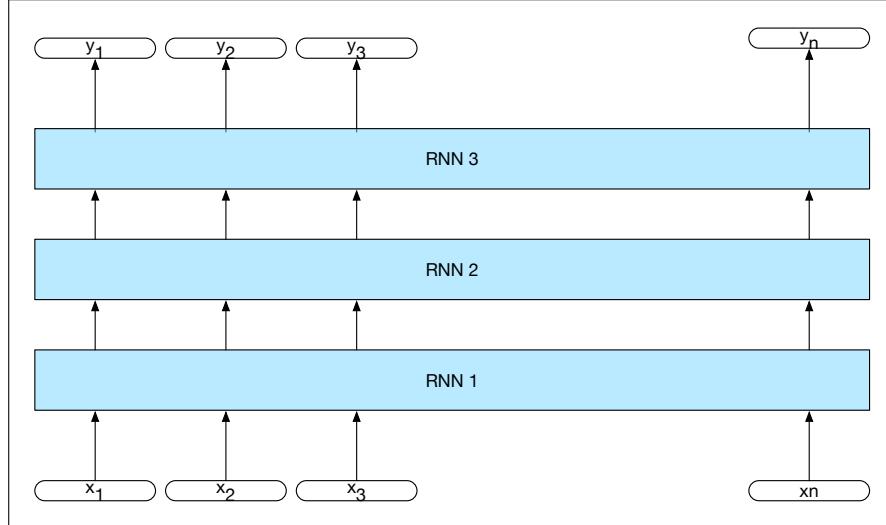


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

It has been demonstrated across numerous tasks that stacked RNNs can outperform single-layer networks. One reason for this success has to do with the network’s ability to *induce representations at differing levels of abstraction across layers*. Just as the early stages of the human visual system detect edges that are then used for finding larger regions and shapes, the initial layers of stacked networks can induce representations that serve as useful abstractions for further layers — representations that might prove difficult to induce in a single RNN.

The optimal number of stacked RNNs is specific to each application and to each training set. However, as the number of stacks is increased the training costs rise quickly.

9.3.2 Bidirectional RNNs

In a simple recurrent network, the hidden state at a given time t represents everything the network knows about the sequence up to that point in the sequence. That is, the hidden state at time t is the result of a function of the inputs from the start up through time t . We can think of this as the context of the network to the left of the current time.

$$h_t^f = \text{RNN}_{\text{forward}}(x_1^t)$$

Where h_t^f corresponds to the normal hidden state at time t , and represents everything the network has gleaned from the sequence to that point.

In many applications we have access to the entire input sequence all at once. We might ask whether it is helpful to take advantage of the context to the right of the current input as well. One way to recover such information is to train an RNN on an input sequence in reverse, using exactly the same kind of networks that we've been discussing. With this approach, the hidden state at time t now represents information about the sequence to the right of the current input.

$$h_t^b = \text{RNN}_{\text{backward}}(x_t^n)$$

Here, the hidden state h_t^b represents all the information we have discerned about the sequence from t to the end of the sequence.

bidirectional RNN

Combining the forward and backward networks results in a **bidirectional RNN** (Schuster and Paliwal, 1997). A Bi-RNN consists of two independent RNNs, one where the input is processed from the start to the end, and the other from the end to the start. We then combine the outputs of the two networks into a single representation that captures both the left and right contexts of an input at each point in time.

$$h_t = h_t^f \oplus h_t^b$$

Fig. 9.11 illustrates a bidirectional network where the outputs of the forward and backward pass are concatenated. Other simple ways to combine the forward and backward contexts include element-wise addition or multiplication. The output at each step in time thus captures information to the left and to the right of the current input. In sequence labeling applications, these concatenated outputs can serve as the basis for a local labeling decision.

Bidirectional RNNs have also proven to be quite effective for sequence classification. Recall from Fig. 9.10, that for sequence classification we used the final hidden state of the RNN as the input to a subsequent feedforward classifier. A difficulty with this approach is that the final state naturally reflects more information about the end of the sentence than its beginning. Bidirectional RNNs provide a simple solution to this problem; as shown in Fig. 9.12, we simply combine the final hidden states from the forward and backward passes and use that as input for follow-on processing. Again, concatenation is a common approach to combining the two outputs but element-wise summation, multiplication or averaging are also used.

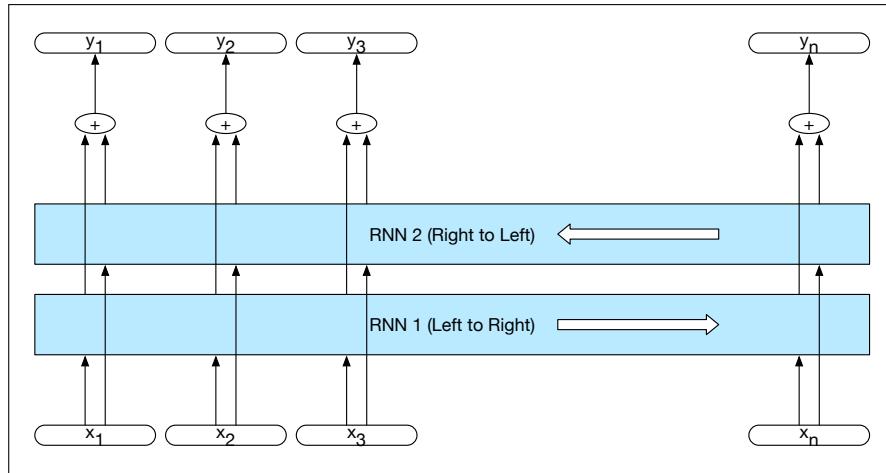


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions with the output of each model at each time point concatenated to represent the state of affairs at that point in time. The box wrapped around the forward and backward network emphasizes the modular nature of this architecture.

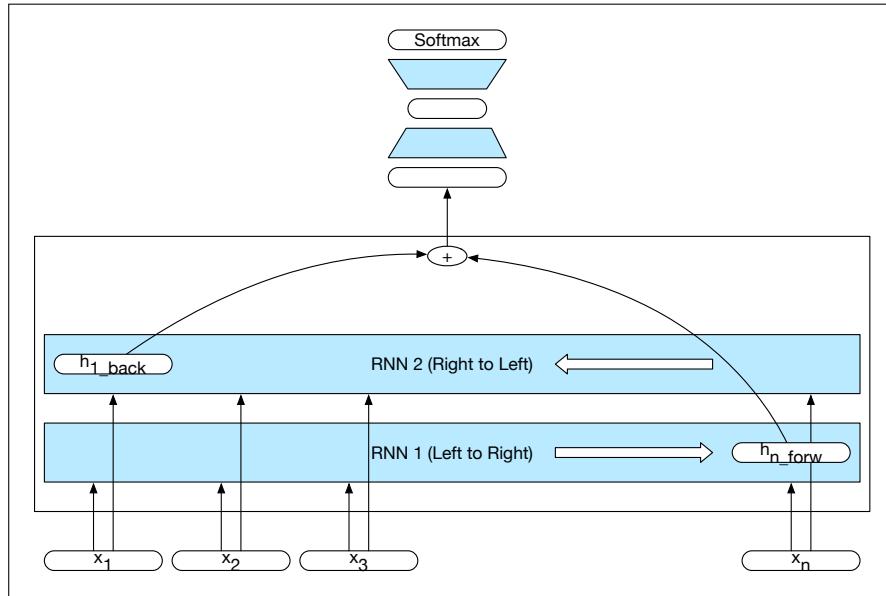


Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

9.4 Managing Context in RNNs: LSTMs and GRUs

In practice, it is quite difficult to train RNNs for tasks that require a network to make use of information distant from the current point of processing. Despite having access to the entire preceding sequence, the information encoded in hidden states tends to be fairly local, more relevant to the most recent parts of the input sequence and recent decisions. It is often the case, however, that distant information is critical

to many language applications. To see this, consider the following example in the context of language modeling.

(9.15) The flights the airline was cancelling were full.

Assigning a high probability to *was* following *airline* is straightforward since *airline* provides a strong local context for the singular agreement. However, assigning an appropriate probability to *were* is quite difficult, not only because the plural *flights* is quite distant, but also because the intervening context involves singular constituents. Ideally, a network should be able to retain the distant information about plural *flights* until it is needed, while still processing the intermediate parts of the sequence correctly.

One reason for the inability of RNNs to carry forward critical information is that the hidden layers, and, by extension, the weights that determine the values in the hidden layer, are being asked to perform two tasks simultaneously: provide information useful for the current decision, and updating and carrying forward information required for future decisions.

vanishing gradients

A second difficulty with training SRNs arises from the need to backpropagate the error signal back through time. Recall from Section 9.1.2 that the hidden layer at time t contributes to the loss at the next time step since it takes part in that calculation. As a result, during the backward pass of training, the hidden layers are subject to repeated multiplications, as determined by the length of the sequence. A frequent result of this process is that the gradients are eventually driven to zero – the so-called **vanishing gradients** problem.

To address these issues, more complex network architectures have been designed to explicitly manage the task of maintaining relevant context over time. More specifically, the network needs to learn to forget information that is no longer needed and to remember information required for decisions still to come.

9.4.1 Long Short-Term Memory

Long short-term memory

Long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) divide the context management problem into two sub-problems: removing information no longer needed from the context, and adding information likely to be needed for later decision making. The key to solving both problems is to learn how to manage this context rather than hard-coding a strategy into the architecture. LSTMs accomplish this by first adding an explicit context layer to the architecture (in addition to the usual recurrent hidden layer), and through the use of specialized neural units that make use of *gates* to control the flow of information into and out of the units that comprise the network layers. These gates are implemented through the use of additional weights that operate sequentially on the input, and previous hidden layer, and previous context layers.

The gates in an LSTM share a common design pattern; each consists of a feed-forward layer, followed by a sigmoid activation function, followed by a pointwise multiplication with the layer being gated. The choice of the sigmoid as the activation function arises from its tendency to push its outputs to either 0 or 1. Combining this with a pointwise multiplication has an effect similar to that of a binary mask. Values in the layer being gated that align with values near 1 in the mask are passed through nearly unchanged; values corresponding to lower values are essentially erased.

forget gate

The first gate we'll consider is the **forget gate**. The purpose of this gate to delete information from the context that is no longer needed. The forget gate computes a weighted sum of the previous state's hidden layer and the current input and passes

that through a sigmoid. This mask is then multiplied by the context vector to remove the information from context that is no longer required.

$$\begin{aligned} f_t &= \sigma(U_f h_{t-1} + W_f x_t) \\ k_t &= c_{t-1} \odot f_t \end{aligned}$$

The next task is compute the actual information we need to extract from the previous hidden state and current inputs — the same basic computation we've been using for all our recurrent networks.

$$g_t = \tanh(U_g h_{t-1} + W_g x_t) \quad (9.16)$$

add gate Next, we generate the mask for the **add gate** to select the information to add to the current context.

$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \quad (9.17)$$

$$j_t = g_t \odot i_t \quad (9.18)$$

Next, we add this to the modified context vector to get our new context vector.

$$c_t = j_t + k_t \quad (9.19)$$

output gate The final gate we'll use is the **output gate** which is used to decide what information is required for the current hidden state (as opposed to what information needs to be preserved for future decisions).

$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \quad (9.20)$$

$$h_t = o_t \odot \tanh(c_t) \quad (9.21)$$

$$(9.22)$$

Fig. 9.13 illustrates the complete computation for a single LSTM unit. Given the appropriate weights for the various gates, an LSTM accepts as input the context layer, and hidden layer from the previous time step, along with the current input vector. It then generates updated context and hidden vectors as output. The hidden layer, h_t , can be used as input to subsequent layers in a stacked RNN, or to generate an output for the final layer of a network.

9.4.2 Gated Recurrent Units

LSTMs introduce a considerable number of additional parameters to our recurrent networks. We now have 8 sets of weights to learn (i.e., the U and W for each of the 4 gates within each unit), whereas with simple recurrent units we only had 2. Training these additional parameters imposes a much significantly higher training cost. Gated Recurrent Units (GRUs)(Cho et al., 2014) ease this burden by dispensing with the use of a separate context vector, and by reducing the number of gates to 2 — a reset gate, r and an update gate, z .

$$r_t = \sigma(U_r h_{t-1} + W_r x_t) \quad (9.23)$$

$$z_t = \sigma(U_z h_{t-1} + W_z x_t) \quad (9.24)$$

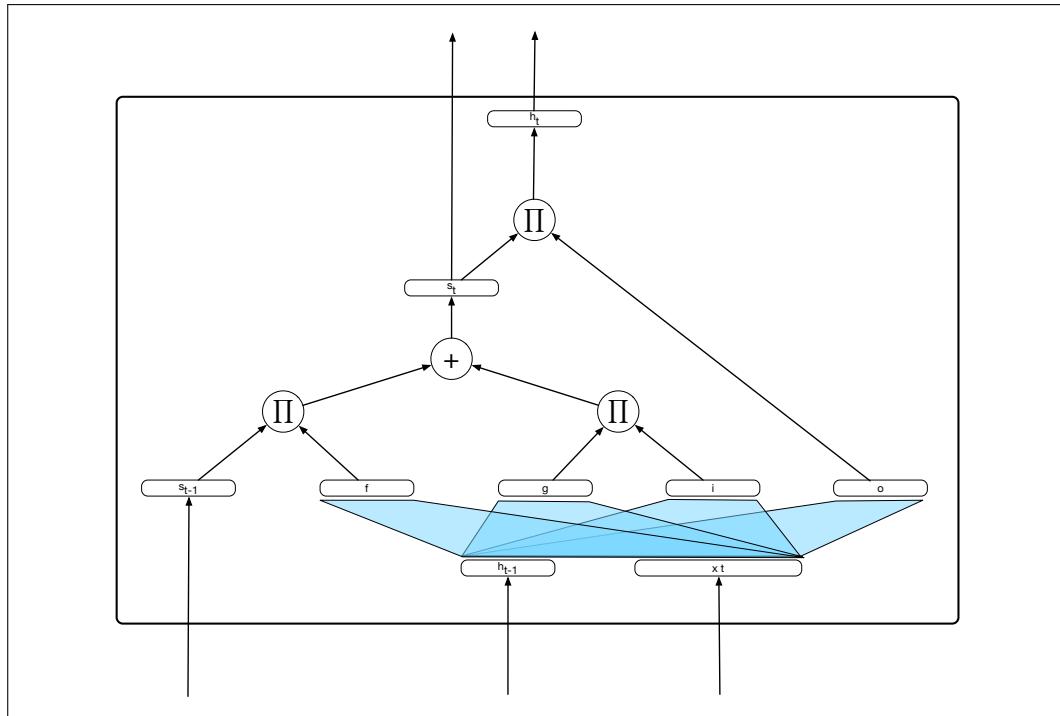


Figure 9.13 A single LSTM unit displayed as a computation graph. The inputs to each unit consists of the current input, x , the previous hidden state, h_{t-1} , and the previous context, c_{t-1} . The outputs are a new hidden state, h_t and an updated context, c_t .

As with LSTMs, the use of the sigmoid in the design of these gates results in a binary-like mask that either blocks information with values near zero or allows information to pass through unchanged with values near one. The purpose of the reset gate is to decide which aspects of the previous hidden state are relevant to the current context and what can be ignored. This is accomplished by performing an element-wise multiplication of r with the value of the previous hidden state. We then use this masked value in computing an intermediate representation for the new hidden state at time t .

$$\tilde{h}_t = \tanh(U(r_t \odot h_{t-1}) + Wx_t) \quad (9.25)$$

The job of the update gate z is to determine which aspects of this new state will be used directly in the new hidden state and which aspects of the previous state need to be preserved for future use. This is accomplished by using the values in z to interpolate between the old hidden state and the new one.

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t \quad (9.26)$$

9.4.3 Gated Units, Layers and Networks

The neural units used in LSTMs and GRUs are obviously much more complex than those used in basic feedforward networks. Fortunately, this complexity is encapsulated within the basic processing units, allowing us to maintain modularity and to

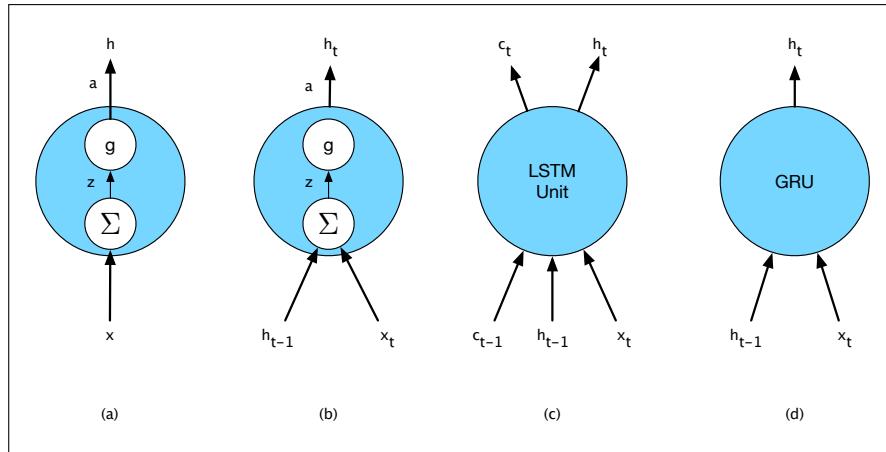


Figure 9.14 Basic neural units used in feedforward, simple recurrent networks (SRN), long short-term memory (LSTM) and gate recurrent units.

easily experiment with different architectures. To see this, consider Fig. 9.14 which illustrates the inputs and outputs associated with each kind of unit.

At the far left, (a) is the basic feedforward unit where a single set of weights and a single activation function determine its output, and when arranged in a layer there are no connections among the units in the layer. Next, (b) represents the unit in a simple recurrent network. Now there are two inputs and an additional set of weights to go with it. However, there is still a single activation function and output.

The increased complexity of the LSTM (c) and GRU (d) units on the right is encapsulated within the units themselves. The only additional external complexity for the LSTM over the basic recurrent unit (b) is the presence of the additional context vector as an input and output. The GRU units have the same input and output architecture as the simple recurrent unit.

This modularity is key to the power and widespread applicability of LSTM and GRU units. LSTM and GRU units can be substituted into any of the network architectures described in Section 9.3. And, as with simple RNNs, multi-layered networks making use of gated units can be unrolled into deep feedforward networks and trained in the usual fashion with backpropagation.

9.5 Words, Subwords and Characters

To this point, we've been assuming that the inputs to our networks would be word embeddings. As we've seen, word-based embeddings are great at capturing distributional (syntactic and semantic) similarity between words. However, there are drawbacks to an exclusively word-based approach:

- For some languages and applications, the lexicon is simply too large to practically represent every possible word as an embedding. Some means of composing words from smaller bits is needed.
- No matter how large the lexicon, we will always encounter unknown words due to new words entering the language, misspellings and borrowings from other languages.

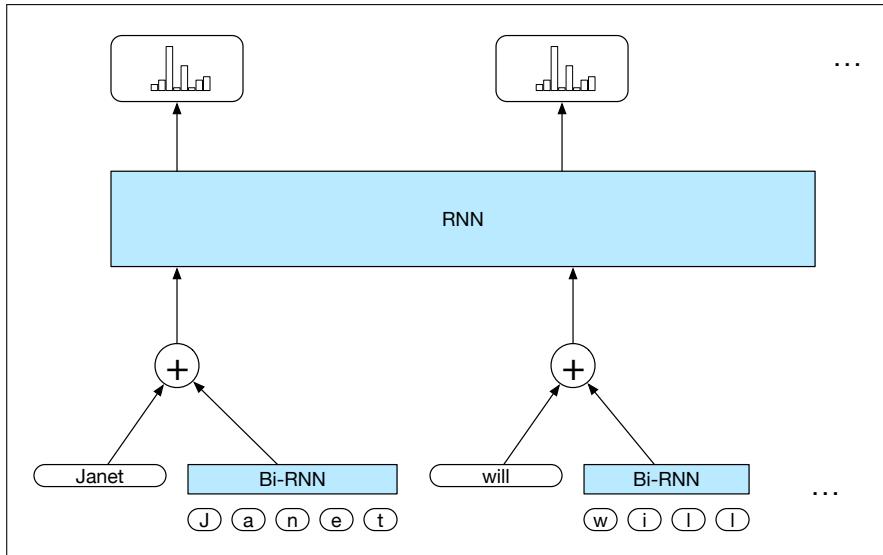


Figure 9.15 Sequence labeling RNN that accepts distributional word embeddings augmented with character-level word embeddings.

- Morphological information, below the word level, is a critical source of information for many languages and many applications. Word-based methods are blind to such regularities.

A wide variety of alternatives to the word-based approach have been explored over the past few years. The following are among the primary approaches that have been tried.

- Ignore words altogether and simply use character sequences as the input to RNNs.
- Use subword units such as those derived from byte-pair encoding or phonetic analysis as inputs.
- Use full-blown morphological analysis to derive a linguistically motivated input sequence.

Perhaps not surprisingly there is no clear one-best approach for all applications for all languages.

One particularly successful approach combines word embeddings with embeddings derived from the characters that make up the words. Fig. 9.15 illustrates an approach in the context of part-of-speech tagging. The upper part of the diagram consists of an RNN that accepts an input sequence and outputs a softmax distribution over the tags for each element of the input. Note that this RNN can be arbitrarily complex, consisting of stacked and/or bidirectional network layers.

The inputs to this network consist of ordinary word embeddings enriched with character-level information. Specifically, each input consists of the concatenation of the normal word embedding with embeddings derived from a bidirectional RNN that accepts the character sequences for each word as input, as shown in the lower part of the figure.

The character sequence for each word in the input is run through a bidirectional RNN consisting of two independent RNNs — one that processes the sequence left-to-right and the other right-to-left. As discussed in Section 9.3.2, the final hidden states of the left-to-right and right-to-left networks are concatenated to represent the

composite character-level representation of each word. Critically, these character embeddings are trained in the context of the overall task; the loss from the part-of-speech softmax layer is propagated all the way back to the character embeddings.

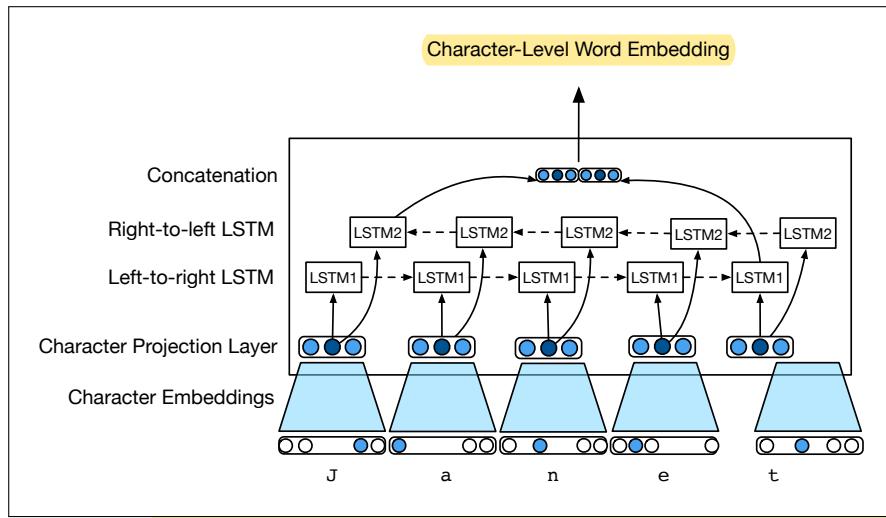


Figure 9.16 Bi-RNN accepts word character sequences and emits embeddings derived from a forward and backward pass over the sequence. The network itself is trained in the context of a larger end-application where the loss is propagated all the way through to the character vector embeddings.

9.6 Summary

This chapter has introduced the concept of recurrent neural networks and how they can be applied to language problems. Here's a summary of the main points that we covered:

- In simple Recurrent Neural Networks sequences are processed naturally as an element at a time.
- The output of a neural unit at a particular point in time is based both on the current input and value of the hidden layer from the previous time step.
- RNNs can be trained with a straightforward extension of the backpropagation algorithm, known as backpropagation through time (BPTT).
- Common language-based applications for RNNs include:
 - Probabilistic language modeling, where the model assigns a probability to a sequence, or to the next element of a sequence given the preceding words.
 - Auto-regressive generation using a trained language model.
 - Sequence labeling, where each element of a sequence is assigned a label, as with part-of-speech tagging.
 - Sequence classification, where an entire text is assigned to a category, as in spam detection, sentiment analysis or topic classification.
- Simple recurrent networks often fail since it is extremely difficult to successfully train them due to problems maintaining useful gradients over time.

- More complex gated architectures such as LSTMs and GRUs are designed to overcome these issues by explicitly managing the task of deciding what to remember and forget in their hidden and context layers.

Bibliographical and Historical Notes

Influential investigations of the kind of simple RNNs discussed here were conducted in the context of the Parallel Distributed Processing (PDP) group at UC San Diego in the 1980’s. Much of this work was directed at human cognitive modeling rather than practical NLP applications Rumelhart et al. 1986 McClelland et al. 1986. Models using recurrence at the hidden layer in a feedforward network (Elman networks) were introduced by Elman (1990). Similar architectures were investigated by Jordan (1986) with a recurrence from the output layer, and Mathis and Mozer (1995) with the addition of a recurrent context layer prior to the hidden layer. The possibility of unrolling a recurrent network into an equivalent feedforward network is discussed in (Rumelhart et al., 1986).

In parallel with work in cognitive modeling, RNNs were investigated extensively in the continuous domain in the signal processing and speech communities (Giles et al., 1994). Schuster and Paliwal (1997) introduced bidirectional RNNs and described results on the TIMIT phoneme transcription task.

While theoretically interesting, the difficulty with training RNNs and managing context over long sequences impeded progress on practical applications. This situation changed with the introduction of LSTMs in Hochreiter and Schmidhuber (1997). Impressive performance gains were demonstrated on tasks at the boundary of signal processing and language processing including phoneme recognition (Graves and Schmidhuber, 2005), handwriting recognition (Graves et al., 2007) and most significantly speech recognition (Graves et al., 2013).

Interest in applying neural networks to practical NLP problems surged with the work of Collobert and Weston (2008) and Collobert et al. (2011). These efforts made use of learned word embeddings, convolutional networks, and end-to-end training. They demonstrated near state-of-the-art performance on a number of standard shared tasks including part-of-speech tagging, chunking, named entity recognition and semantic role labeling without the use of hand-engineered features.

Approaches that married LSTMs with pre-trained collections of word-embeddings based on word2vec (Mikolov et al., 2013) and GLOVE (Pennington et al., 2014), quickly came to dominate many common tasks: part-of-speech tagging (Ling et al., 2015a), syntactic chunking (Søgaard and Goldberg, 2016), and named entity recognition via IOB tagging Chiu and Nichols 2016, Ma and Hovy 2016, opinion mining (Irsoy and Cardie, 2014), semantic role labeling (Zhou and Xu, 2015a) and AMR parsing (Foland and Martin, 2016). As with the earlier surge of progress involving statistical machine learning, these advances were made possible by the availability of training data provided by CONLL, SemEval, and other shared tasks, as well as shared resources such as Ontonotes (Pradhan et al., 2007b), and PropBank (Palmer et al., 2005).

Encoder-Decoder Models, Attention, and Contextual Embeddings

It is all well and good to copy what one sees, but it is much better to draw only what remains in one's memory. This is a transformation in which imagination and memory collaborate.

Edgar Degas

In Chapter 9 we explored recurrent neural networks along with some of their common use cases, including language modeling, contextual generation, and sequence labeling. A common thread in these applications was the notion of transduction — input sequences being transformed into output sequences in a one-to-one fashion. Here, we'll explore an approach that extends these models and provides much greater flexibility across a range of applications. Specifically, we'll introduce **encoder-decoder** networks, or **sequence-to-sequence** models, that are capable of generating contextually appropriate, arbitrary length, output sequences. Encoder-decoder networks have been applied to a very wide range of applications including machine translation, summarization, question answering, and dialogue modeling.

The key idea underlying these networks is the use of an **encoder** network that takes an input sequence and creates a contextualized representation of it. This representation is then passed to a **decoder** which generates a task-specific output sequence. The encoder and decoder networks are typically implemented with the same architecture, often using recurrent networks of the kind we studied in Chapter 9. And as with the deep networks introduced there, the encoder-decoder architecture allows networks to be trained in an end-to-end fashion for each application.

10.1 Neural Language Models and Generation Revisited

To understand the design of encoder-decoder networks let's return to neural language models and the notion of autoregressive generation. Recall that in a simple recurrent network, the value of the hidden state at a particular point in time is a function of the previous hidden state and the current input; the network output is then a function of this new hidden state.

$$\begin{aligned} h_t &= g(Uh_{t-1} + Wx_t) \\ y_t &= f(Vh_t) \end{aligned}$$

Here, U , V , and W are weight matrices which are adjusted during training, g is a suitable non-linear activation function such as *tanh* or ReLU, and in the common case of classification f is a softmax over the set of possible outputs. In practice, gated networks using LSTMs or GRUs are used in place of these simple RNNs. To

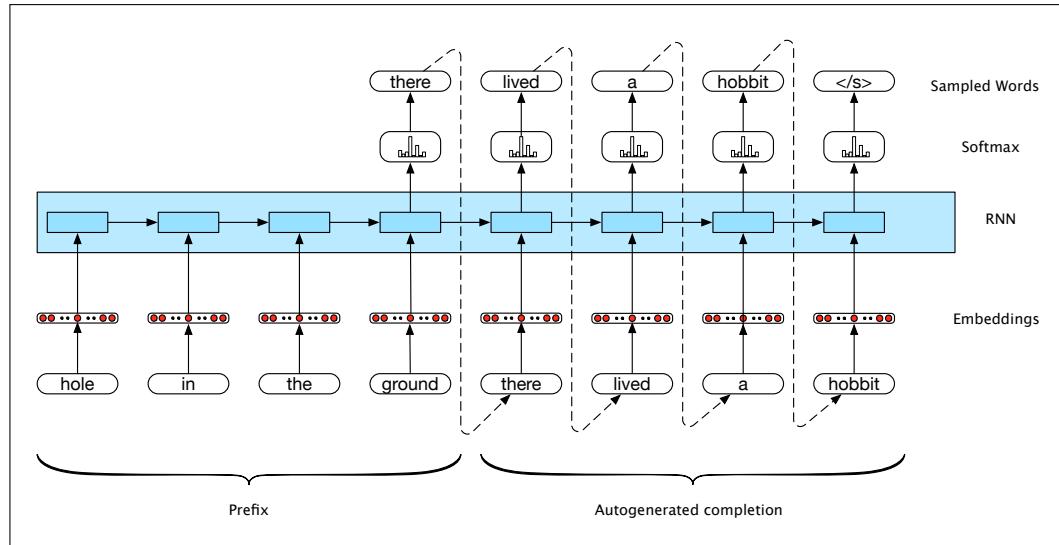


Figure 10.1 Using an RNN to generate the completion of an input phrase.

reflect this, we'll abstract away from the details of the specific RNN being used and simply specify the inputs on which the computation is being based. So the earlier equations will be expressed as follows with the understanding that there is a suitable RNN underneath.

$$\begin{aligned} h_t &= g(h_{t-1}, x_t) \\ y_t &= f(h_t) \end{aligned}$$

To create an RNN-based language model, we train the network to predict the next word in a sequence using a corpus of representative text. Language models trained in this fashion are referred to as autoregressive models. Given a trained model, we can ask the network to generate novel sequences by first randomly sampling an appropriate word as the beginning of a sequence. We then condition the generation of subsequent words on the hidden state from the previous time step as well as the embedding for the word just generated, again sampling from the distribution provided by the softmax. More specifically, during generation the softmax output at each point in time provides us with the probability of every word in the vocabulary given the preceding context, that is $P(y_i|y_{<i}) \forall i \in V$; we then sample a particular word, \hat{y}_i , from this distribution and condition subsequent generation on it. The process continues until the end of sentence token $<\text{s}>$ is generated.

Now, let's consider a simple variation on this scheme. Instead of having the language model generate a sentence from scratch, we have it complete a sequence given a specified prefix. More specifically, we first pass the specified prefix through the language model using forward inference to produce a sequence of hidden states, ending with the hidden state corresponding to the last word of the prefix. We then begin generating as we did earlier, but using the final hidden state of the prefix as our starting point. The result of this process is a novel output sequence that should be a reasonable completion of the prefix input.

Fig. 10.1 illustrates this basic scheme. The portion of the network on the left processes the provided prefix, while the right side executes the subsequent autoregressive generation. Note that the goal of the lefthand portion of the network is to generate a series of hidden states from the given input; there are no outputs

associated with this part of the process until we reach the end of the prefix.

Now, consider an ingenious extension of this idea from the world of machine translation (MT), the task of automatically translating sentences from one language into another. The primary resources used to train modern translation systems are known as parallel texts, or **bitexts**. These are large text collections consisting of pairs of sentences from different languages that are translations of one another. Traditionally in MT, the text being translated is referred to as the **source** and the translation output is called the **target**.

To extend language models and autoregressive generation to machine translation, we'll first add an end-of-sentence marker at the end of each bitext's source sentence and then simply concatenate the corresponding target to it. These concatenated source-target pairs can now serve as training data for a combined language model. Training proceeds as with any RNN-based language model. The network is trained autoregressively to predict the next word in a set of sequences comprised of the concatenated source-target bitexts, as shown in Fig. 10.2.

To translate a source text using the trained model, we run it through the network performing forward inference to generate hidden states until we get to the end of the source. Then we begin autoregressive generation, asking for a word in the context of the hidden layer from the end of the source input as well as the end-of-sentence marker. Subsequent words are conditioned on the previous hidden state and the embedding for the last word generated.

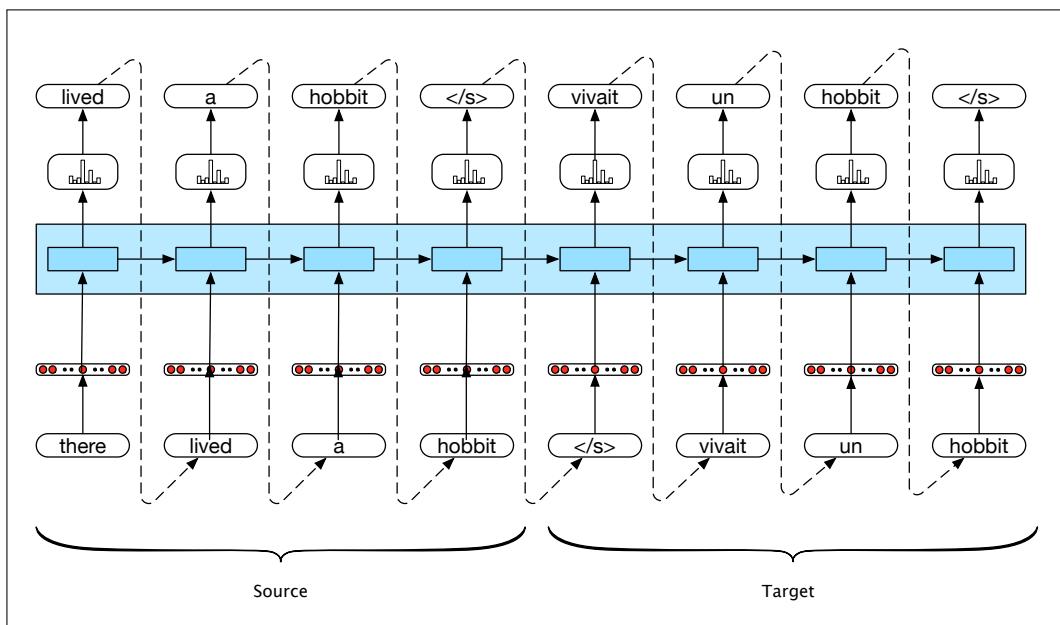


Figure 10.2 Training setup for a neural language model approach to machine translation. Source-target bitexts are concatenated and used to train a language model.

Early efforts using this clever approach demonstrated surprisingly good results on standard datasets and led to a series of innovations that were the basis for networks discussed in the remainder of this chapter. Chapter 11 provides an in-depth discussion of the fundamental issues in translation as well as the current state-of-the-art approaches to MT. Here, we'll focus on the powerful models that arose from these early efforts.

10.2 Encoder-Decoder Networks

encoder-decoder

Fig. 10.3 abstracts away from the specifics of machine translation and illustrates a basic **encoder-decoder** architecture. The elements of the network on the left process the input sequence and comprise the **encoder**, the entire purpose of which is to generate a contextualized representation of the input. In this network, this representation is embodied in the final hidden state of the encoder, h_n , which in turn feeds into the first hidden state of the decoder. The **decoder** network on the right takes this state and autoregressively generates a sequence of outputs.

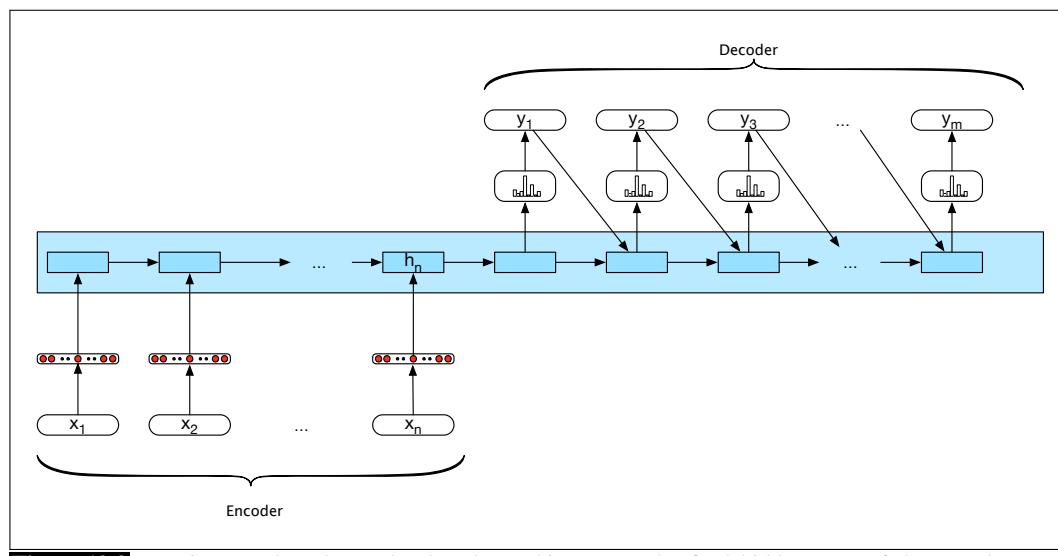


Figure 10.3 Basic RNN-based encoder-decoder architecture. The final hidden state of the encoder RNN serves as the context for the decoder in its role as h_0 in the decoder RNN.

This basic architecture is consistent with the original applications of neural models to machine translation. However, it embodies a number of design choices that are less than optimal. Among the major ones are that the encoder and the decoder are assumed to have the same internal structure (RNNs in this case), that the final state of the encoder is the only context available to the decoder, and finally that this context is only available to the decoder as its initial hidden state. Abstracting away from these choices, we can say that encoder-decoder networks consist of three components:

1. An **encoder** that accepts an input sequence, x_1^n , and generates a corresponding sequence of contextualized representations, h_1^n .
2. A **context vector**, c , which is a function of h_1^n , and conveys the essence of the input to the decoder.
3. And a **decoder**, which accepts c as input and generates an arbitrary length sequence of hidden states h_1^m , from which a corresponding sequence of output states y_1^m , can be obtained.

Fig. 10.4 illustrates this abstracted architecture. Let's now explore some of the possibilities for each of the components.

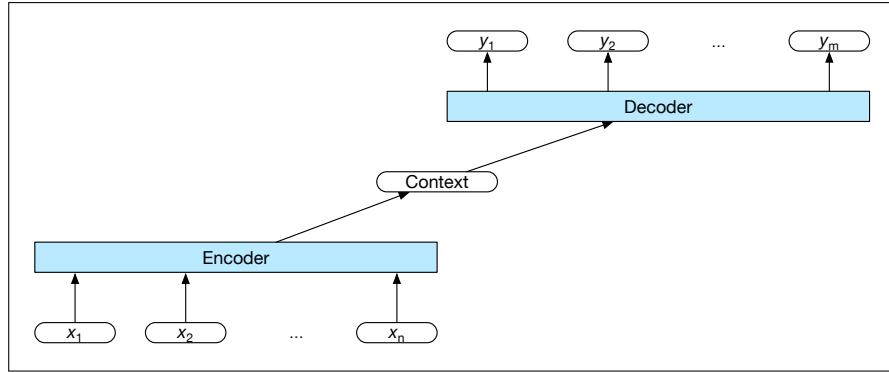


Figure 10.4 Basic architecture for an abstract encoder-decoder network. The context is a function of the vector of contextualized input representations and may be used by the decoder in a variety of ways.

Encoder

Simple RNNs, LSTMs, GRUs, convolutional networks, as well as transformer networks (discussed later in this chapter), can all be employed as encoders. For simplicity, our figures show only a single network layer for the encoder, however, stacked architectures are the norm, where the output states from the top layer of the stack are taken as the final representation. A widely used encoder design makes use of stacked Bi-LSTMs where the hidden states from top layers from the forward and backward passes are concatenated as described in Chapter 9 to provide the contextualized representations for each time step.

Decoder

For the decoder, autoregressive generation is used to produce an output sequence, an element at a time, until an end-of-sequence marker is generated. This incremental process is guided by the context provided by the encoder as well as any items generated for earlier states by the decoder. Again, a typical approach is to use an LSTM or GRU-based RNN where the context consists of the final hidden state of the encoder, and is used to initialize the first hidden state of the decoder. (To help keep things straight, we'll use the superscripts e and d where needed to distinguish the hidden states of the encoder and the decoder.) Generation proceeds as described earlier where each hidden state is conditioned on the previous hidden state and output generated in the previous state.

$$\begin{aligned} c &= h_n^e \\ h_0^d &= c \end{aligned}$$

$$\begin{aligned} h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d) \\ z_t &= f(h_t^d) \\ y_t &= \text{softmax}(z_t) \end{aligned}$$

Recall, that g is a stand-in for some flavor of RNN and \hat{y}_{t-1} is the embedding for the output sampled from the softmax at the previous step.

A weakness of this approach is that the context vector, c , is only directly available at the beginning of the process and its influence will wane as the output sequence is generated. A solution is to make the context vector c available at each step

in the decoding process by adding it as a parameter to the computation of the current hidden state.

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$

A common approach to the calculation of the output layer y is to base it solely on this newly computed hidden state. While this cleanly separates the underlying recurrence from the output generation task, it makes it difficult to keep track of what has already been generated and what hasn't. An alternative approach is to condition the output on both the newly generated hidden state, the output generated at the previous state, and the encoder context.

$$y_t = \text{softmax}(\hat{y}_{t-1}, z_t, c)$$

Finally, as shown earlier, the output y at each time consists of a softmax computation over the set of possible outputs (the vocabulary in the case of language models). What one does with this distribution is task-dependent, but it is critical since the recurrence depends on choosing a particular output, \hat{y} , from the softmax to condition the next step in decoding. We've already seen several of the possible options for this. For neural generation, where we are trying to generate novel outputs, we can simply sample from the softmax distribution. However, for applications like MT where we're looking for a specific output sequence, random sampling isn't appropriate and would likely lead to some strange output. An alternative is to choose the most likely output at each time step by taking the argmax over the softmax output:

$$\hat{y} = \text{argmax}_i P(y_i | y_{<i})$$

This is easy to implement but as we've seen several times with sequence labeling, independently choosing the argmax over a sequence is not a reliable way of arriving at a good output since it doesn't guarantee that the individual choices being made make sense together and combine into a coherent whole. With sequence labeling we addressed this with a CRF-layer over the output token types combined with a Viterbi-style dynamic programming search. Unfortunately, this approach is not viable here since the dynamic programming invariant doesn't hold.

Beam Search

Beam Search

A viable alternative is to view the decoding problem as a heuristic state-space search and systematically explore the space of possible outputs. The key to such an approach is controlling the exponential growth of the search space. To accomplish this, we'll use a technique called **beam search**. Beam search operates by combining a breadth-first-search strategy with a heuristic filter that scores each option and prunes the search space to stay within a fixed-size memory footprint, called the beam width.

At the first step of decoding, we select the B -best options from the softmax output y , where B is the size of the beam. Each option is scored with its corresponding probability from the softmax output of the decoder. These initial outputs constitute the search frontier. We'll refer to the sequence of partial outputs generated along these search paths as **hypotheses**.

At subsequent steps, each hypothesis on the frontier is extended incrementally by being passed to distinct decoders, which again generate a softmax over the entire vocabulary. To provide the necessary inputs for the decoders, each hypothesis must include not only the words generated thus far but also the context vector, and the

hidden state from the previous step. New hypotheses representing every possible extension to the current ones are generated and added to the frontier. Each of these new hypotheses is scored using $P(y_i|y_{<i})$, which is the product of the probability of current word choice multiplied by the probability of the path that led to it. To control the exponential growth of the frontier, it is pruned to contain only the top B hypotheses.

This process continues until a $\langle \text{S} \rangle$ is generated indicating that a complete candidate output has been found. At this point, the completed hypothesis is removed from the frontier and the size of the beam is reduced by one. The search continues until the beam has been reduced to 0. Leaving us with B hypotheses to consider. Fig. 10.5 illustrates this process with a beam width of 3.

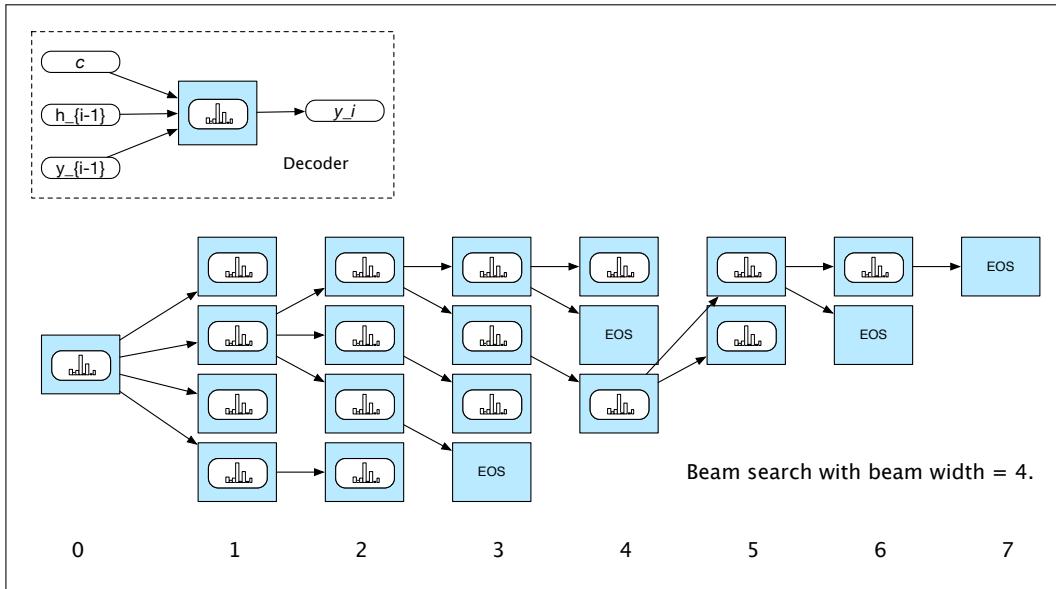


Figure 10.5 Beam decoding with a beam width of 4. At the initial step, the frontier is filled with the best 4 options from the initial state of the decoder. In a breadth-first fashion, each state on the frontier is passed to a decoder which computes a softmax over the entire vocabulary and attempts to enter each as a new state into the frontier subject to the constraint that they are better than the worst state already there. As completed sequences are discovered they are recorded and removed from the frontier and the beam width is reduced by 1.

One final complication arises from the fact that the completed hypotheses may have different lengths. Unfortunately, due to the probabilistic nature of our scoring scheme, longer hypotheses will naturally look worse than shorter ones just based on their length. This was not an issue during the earlier steps of decoding; due to the breadth-first nature of beam search all the hypotheses being compared had the same length. The usual solution to this is to apply some form of length normalization to each of the hypotheses. With normalization, we have B hypotheses and can select the best one, or we can pass all or a subset of them on to a downstream application with their respective scores.

Context

We've defined the context vector c as a function of the hidden states of the encoder, that is, $c = f(h_1^n)$. Unfortunately, the number of hidden states varies with the size of the input, making it difficult to just use them directly as a context for the decode. The basic approach described earlier avoids this issue since c is just the final hidden state

```

function BEAMDECODE(c, beam_width) returns best paths
    y0, h0  $\leftarrow$  0
    path  $\leftarrow$  ()
    complete_paths  $\leftarrow$  ()
    state  $\leftarrow$  (c, y0, h0, path) ;initial state
    frontier  $\leftarrow$  ⟨state⟩ ;initial frontier

    while frontier contains incomplete paths and beamwidth  $>$  0
        extended_frontier  $\leftarrow$  ⟨⟩
        for each state  $\in$  frontier do
            y  $\leftarrow$  DECODE(state)
            for each word i  $\in$  Vocabulary do
                successor  $\leftarrow$  NEWSTATE(state, i, yi)
                new_agenda  $\leftarrow$  ADDTOBEAM(successor, extended_frontier, beam_width)

            for each state in extended_frontier do
                if state is complete do
                    complete_paths  $\leftarrow$  APPEND(complete_paths, state)
                    extended_frontier  $\leftarrow$  REMOVE(extended_frontier, state)
                    beam_width  $\leftarrow$  beam_width - 1
                frontier  $\leftarrow$  extended_frontier

            return completed_paths

function NEWSTATE(state, word, word_prob) returns new state
function ADDTOBEAM(state, frontier, width) returns updated frontier
    if LENGTH(frontier)  $<$  width then
        frontier  $\leftarrow$  INSERT(state, frontier)
    else if SCORE(state)  $>$  SCORE(WORSTOF(frontier))
        frontier  $\leftarrow$  REMOVE(WORSTOF(frontier))
        frontier  $\leftarrow$  INSERT(state, frontier)
    return frontier

```

Figure 10.6 Beam search decoding.

of the encoder. This approach has the advantage of being simple and of reducing the context to a fixed length vector. However, this final hidden state inevitably is more focused on the latter parts of input sequence, rather than the input as whole.

One solution to this problem is to use Bi-RNNs, where the context can be a function of the end state of both the forward and backward passes. As described in Chapter 9, a straightforward approach is to concatenate the final states of the forward and backward passes. An alternative is to simply sum or average the encoder hidden states to produce a context vector. Unfortunately, this approach loses useful information about each of the individual encoder states that might prove useful in decoding.

10.3 Attention

attention mechanism

To overcome the deficiencies of these simple approaches to context, we'll need a mechanism that can take the entire encoder context into account, that dynamically updates during the course of decoding, and that can be embodied in a fixed-size vector. Taken together, we'll refer such an approach as an **attention mechanism**.

Our first step is to replace the static context vector with one that is dynamically derived from the encoder hidden states at each point during decoding. This context vector, c_i , is generated anew with each decoding step i and takes all of the encoder hidden states into account in its derivation. We then make this context available during decoding by conditioning the computation of the current decoder state on it, along with the prior hidden state and the previous output generated by the decoder.

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

The first step in computing c_i is to compute a vector of scores that capture the relevance of each encoder hidden state to the decoder state captured in h_{i-1}^d . That is, at each state i during decoding we'll compute $score(h_{i-1}^d, h_j^e)$ for each encoder state j .

For now, let's assume that this score provides us with a measure of how similar the decoder hidden state is to each encoder hidden state. To implement this similarity score, let's begin with the straightforward approach introduced in Chapter 6 of using the dot product between vectors.

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

The result of the dot product is a scalar that reflects the degree of similarity between the two vectors. And the vector of scores over all the encoder hidden states gives us the relevance of each encoder state to the current step of the decoder.

While the simple dot product can be effective, it is a static measure that does not facilitate adaptation during the course of training to fit the characteristics of given applications. A more robust similarity score can be obtained by parameterizing the score with its own set of weights, W_s .

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s h_j^e$$

By introducing W_s to the score, we are giving the network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

To make use of these scores, we'll next normalize them with a softmax to create a vector of weights, α_{ij} , that tells us the proportional relevance of each encoder hidden state j to the current decoder state, i .

$$\alpha_{ij} = \text{softmax}(score(h_{i-1}^d, h_j^e) \quad \forall j \in e)$$

$$= \frac{\exp(score(h_{i-1}^d, h_j^e))}{\sum_k \exp(score(h_{i-1}^d, h_k^e))}$$

Finally, given the distribution in α , we can compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

$$c_i = \sum_j \alpha_{ij} h_j^e \tag{10.1}$$

With this, we finally have a fixed-length context vector that takes into account information from the entire encoder state that is dynamically update to reflect the needs of the decoder at each step of decoding. Fig. 10.7 illustrates an encoder-decoder network with attention.

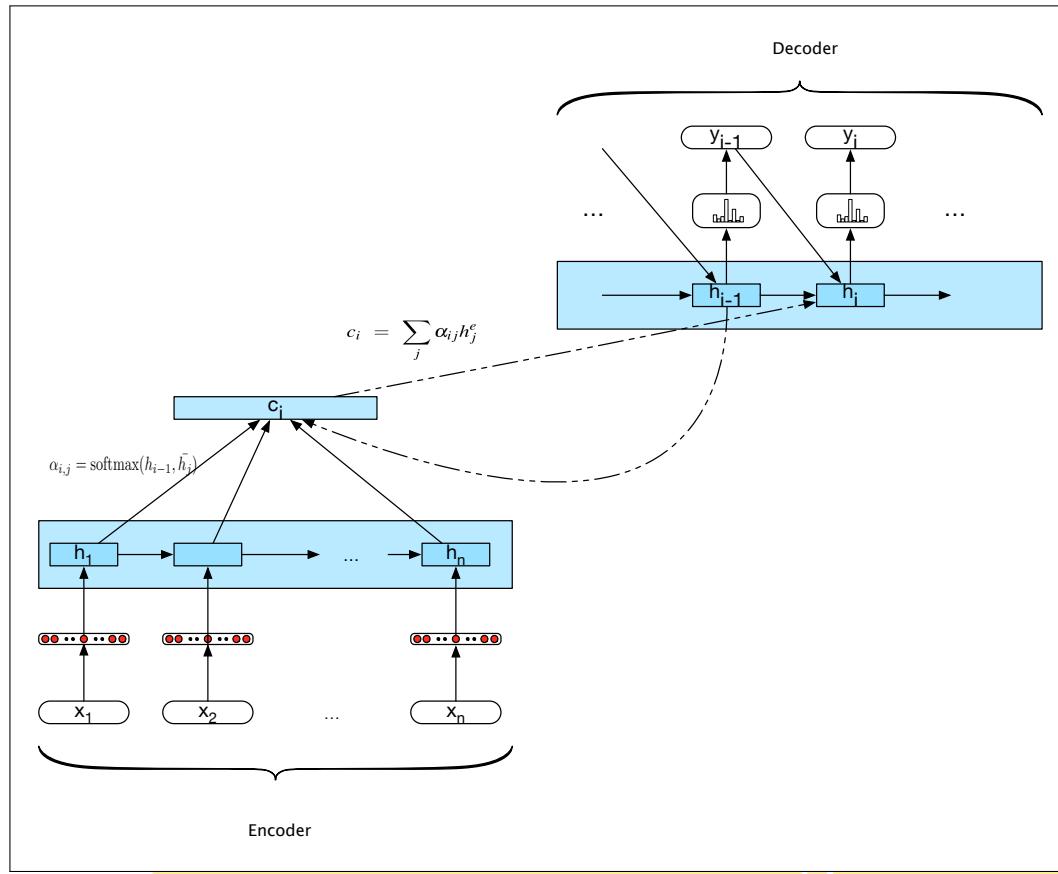


Figure 10.7 Encoder-decoder network with attention. Computing the value for h_i is based on the previous hidden state, the previous word generated, and the current context vector c_i . This context vector is derived from the attention computation based on comparing the previous hidden state to all of the encoder hidden states.

10.4 Applications of Encoder-Decoder Networks

The addition of attention to basic encoder-decoder networks led to rapid improvement in performance across a wide-range of applications including summarization, sentence simplification, question answering and image captioning.

10.5 Self-Attention and Transformer Networks

10.6 Summary

- Encoder-decoder networks
- Attention
- Transformers

Bibliographical and Historical Notes

CHAPTER

11

Machine Translation

Placeholder

The study of grammar has an ancient pedigree; Panini’s grammar of Sanskrit was written over two thousand years ago and is still referenced today in teaching Sanskrit. Despite this history, knowledge of grammar remains spotty at best. In this chapter, we make a preliminary stab at addressing some of these gaps in our knowledge of grammar and syntax, as well as introducing some of the formal mechanisms that are available for capturing this knowledge in a computationally useful manner.

syntax

The word **syntax** comes from the Greek *sýntaxis*, meaning “setting out together or arrangement”, and refers to the way words are arranged together. We have seen various syntactic notions in previous chapters. The regular languages introduced in Chapter 2 offered a simple way to represent the ordering of strings of words, and Chapter 3 showed how to compute probabilities for these word sequences. Chapter 8 showed that part-of-speech categories could act as a kind of equivalence class for words. In this chapter and the next few we introduce a variety of syntactic phenomena and models for syntax that go well beyond these simpler approaches.

The bulk of this chapter is devoted to the topic of context-free grammars. Context-free grammars are the backbone of many formal models of the syntax of natural language (and, for that matter, of computer languages). As such, they are integral to many computational applications, including grammar checking, semantic interpretation, dialogue understanding, and machine translation. They are powerful enough to express sophisticated relations among the words in a sentence, yet computationally tractable enough that efficient algorithms exist for parsing sentences with them (as we show in Chapter 13). In Chapter 14, we show that adding probability to context-free grammars gives us a powerful model of disambiguation. And in Chapter 17 we show how they provide a systematic framework for semantic interpretation.

The constituency grammars we introduce here, however, are not the only possible formal mechanism for modeling syntax. Chapter 15 will introduce syntactic dependencies, an alternative model that is the core representation for **dependency parsing**. Both constituency and dependency formalisms are important for language processing.

In addition to introducing grammar formalism, this chapter also provides a brief overview of the grammar of English. To illustrate our grammars, we have chosen a domain that has relatively simple sentences, the Air Traffic Information System (ATIS) domain ([Hempill et al., 1990](#)). ATIS systems were an early example of spoken language systems for helping book airline reservations. Users try to book flights by conversing with the system, specifying constraints like *I’d like to fly from Atlanta to Denver*.

12.1 Constituency

The fundamental notion underlying the idea of constituency is that of abstraction — groups of words behaving as single units, or constituents. A significant part of developing a grammar involves discovering the inventory of constituents present in the language.

noun phrase

How do words group together in English? Consider the **noun phrase**, a sequence of words surrounding at least one noun. Here are some examples of noun phrases (thanks to Damon Runyon):

Harry the Horse	a high-class spot such as Mindy's
the Broadway coppers	the reason he comes into the Hot Box
they	three parties from Brooklyn

What evidence do we have that these words group together (or “form constituents”)?

One piece of evidence is that they can all appear in similar syntactic environments, for example, before a verb.

three parties from Brooklyn *arrive*...
a high-class spot such as Mindy's *attracts*...
the Broadway coppers *love*...
they *sit*

But while the whole noun phrase can occur before a verb, this is not true of each of the individual words that make up a noun phrase. The following are not grammatical sentences of English (recall that we use an asterisk (*) to mark fragments that are not grammatical English sentences):

*from *arrive*... *as *attracts*...
*the *is*... *spot *sat*...

Thus, to correctly describe facts about the ordering of these words in English, we must be able to say things like “*Noun Phrases can occur before verbs*”.

preposed postposed

Other kinds of evidence for constituency come from what are called **preposed** or **postposed** constructions. For example, the prepositional phrase *on September seventeenth* can be placed in a number of different locations in the following examples, including at the beginning (preposed) or at the end (postposed):

On September seventeenth, I'd like to fly from Atlanta to Denver
I'd like to fly *on September seventeenth* from Atlanta to Denver
I'd like to fly from Atlanta to Denver *on September seventeenth*

But again, while the entire phrase can be placed differently, the individual words making up the phrase cannot be

*On September, I'd like to fly seventeenth from Atlanta to Denver
*On I'd like to fly September seventeenth from Atlanta to Denver
*I'd like to fly on September from Atlanta to Denver seventeenth

12.2 Context-Free Grammars

The most widely used formal system for modeling constituent structure in English and other natural languages is the **Context-Free Grammar**, or **CFG**. Context-

free grammars are also called **Phrase-Structure Grammars**, and the formalism is equivalent to **Backus-Naur Form**, or **BNF**. The idea of basing a grammar on constituent structure dates back to the psychologist Wilhelm Wundt (1900) but was not formalized until Chomsky (1956) and, independently, Backus (1959).

rules

A context-free grammar consists of a set of **rules** or **productions**, each of which expresses the ways that symbols of the language can be grouped and ordered together, and a **lexicon** of words and symbols. For example, the following productions express that an **NP** (or **noun phrase**) can be composed of either a *ProperNoun* or a determiner (*Det*) followed by a *Nominal*; a *Nominal* in turn can consist of one or more *Nouns*.

$$\begin{aligned} NP &\rightarrow Det \ Nominal \\ NP &\rightarrow ProperNoun \\ Nominal &\rightarrow Noun \mid Nominal \ Noun \end{aligned}$$

Context-free rules can be hierarchically embedded, so we can combine the previous rules with others, like the following, that express facts about the lexicon:

$$\begin{aligned} Det &\rightarrow a \\ Det &\rightarrow the \\ Noun &\rightarrow flight \end{aligned}$$

The symbols that are used in a CFG are divided into two classes. The symbols that correspond to words in the language (“the”, “nightclub”) are called **terminal** symbols; the lexicon is the set of rules that introduce these terminal symbols. The symbols that express abstractions over these terminals are called **non-terminals**. In each context-free rule, the item to the right of the arrow (\rightarrow) is an ordered list of one or more terminals and non-terminals; to the left of the arrow is a single non-terminal symbol expressing some cluster or generalization. Notice that in the lexicon, the non-terminal associated with each word is its lexical category, or part-of-speech, which we defined in Chapter 8.

A CFG can be thought of in two ways: as a device for generating sentences and as a device for assigning a structure to a given sentence. Viewing a CFG as a generator, we can read the \rightarrow arrow as “rewrite the symbol on the left with the string of symbols on the right”.

So starting from the symbol:

NP

we can use our first rule to rewrite *NP* as:

Det Nominal

and then rewrite *Nominal* as:

Det Noun

and finally rewrite these parts-of-speech as:

a flight

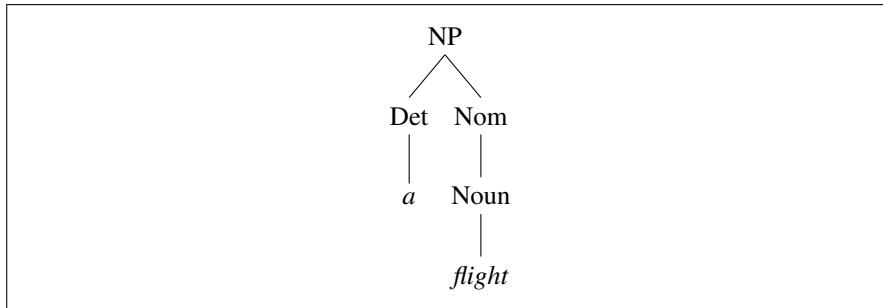
**derivation
parse tree**

We say the string *a flight* can be derived from the non-terminal *NP*. Thus, a CFG can be used to generate a set of strings. This sequence of rule expansions is called a **derivation** of the string of words. It is common to represent a derivation by a **parse tree** (commonly shown inverted with the root at the top). Figure 12.1 shows the tree representation of this derivation.

**dominates
start symbol**

In the parse tree shown in Fig. 12.1, we can say that the node *NP* **dominates** all the nodes in the tree (*Det*, *Nom*, *Noun*, *a*, *flight*). We can say further that it immediately dominates the nodes *Det* and *Nom*.

The formal language defined by a CFG is the set of strings that are derivable from the designated **start symbol**. Each grammar must have one designated start symbol, which is often called *S*. Since context-free grammars are often used to define sentences, *S* is usually interpreted as the “sentence” node, and the set of strings that are derivable from *S* is the set of sentences in some simplified version of English.

**Figure 12.1** A parse tree for “a flight”.

verb phrase Let’s add a few additional rules to our inventory. The following rule expresses the fact that a sentence can consist of a noun phrase followed by a **verb phrase**:

$$S \rightarrow NP\ VP \quad I \text{ prefer a morning flight}$$

A verb phrase in English consists of a verb followed by assorted other things; for example, one kind of verb phrase consists of a verb followed by a noun phrase:

$$VP \rightarrow Verb\ NP \quad \text{prefer a morning flight}$$

Or the verb may be followed by a noun phrase and a prepositional phrase:

$$VP \rightarrow Verb\ NP\ PP \quad \text{leave Boston in the morning}$$

Or the verb phrase may have a verb followed by a prepositional phrase alone:

$$VP \rightarrow Verb\ PP \quad \text{leaving on Thursday}$$

A prepositional phrase generally has a preposition followed by a noun phrase. For example, a common type of prepositional phrase in the ATIS corpus is used to indicate location or direction:

$$PP \rightarrow Preposition\ NP \quad \text{from Los Angeles}$$

The *NP* inside a *PP* need not be a location; *PPs* are often used with times and dates, and with other nouns as well; they can be arbitrarily complex. Here are ten examples from the ATIS corpus:

to Seattle	on these flights
in Minneapolis	about the ground transportation in Chicago
on Wednesday	of the round trip flight on United Airlines
in the evening	of the AP fifty seven flight
on the ninth of July	with a stopover in Nashville

Figure 12.2 gives a sample lexicon, and Fig. 12.3 summarizes the grammar rules we’ve seen so far, which we’ll call \mathcal{L}_0 . Note that we can use the or-symbol | to indicate that a non-terminal has alternate possible expansions.

We can use this grammar to generate sentences of this “ATIS-language”. We start with *S*, expand it to *NP VP*, then choose a random expansion of *NP* (let’s say, to *I*), and a random expansion of *VP* (let’s say, to *Verb NP*), and so on until we generate the string *I prefer a morning flight*. Figure 12.4 shows a parse tree that represents a complete derivation of *I prefer a morning flight*.

bracketed notation

It is sometimes convenient to represent a parse tree in a more compact format called **bracketed notation**; here is the bracketed representation of the parse tree of Fig. 12.4:

<i>Noun</i> → flights breeze trip morning
<i>Verb</i> → is prefer like need want fly
<i>Adjective</i> → cheapest non-stop first latest other direct
<i>Pronoun</i> → me I you it
<i>Proper-Noun</i> → Alaska Baltimore Los Angeles Chicago United American
<i>Determiner</i> → the a an this these that
<i>Preposition</i> → from to on near
<i>Conjunction</i> → and or but

Figure 12.2 The lexicon for \mathcal{L}_0 .

Grammar Rules	Examples
$S \rightarrow NP VP$	I + want a morning flight
$NP \rightarrow Pronoun$	I
Proper-Noun	Los Angeles
Det Nominal	a + flight
$Nominal \rightarrow Nominal\ Noun$	morning + flight
Noun	flights
$VP \rightarrow Verb$	do
Verb NP	want + a flight
Verb NP PP	leave + Boston + in the morning
Verb PP	leaving + on Thursday
$PP \rightarrow Preposition\ NP$	from + Los Angeles

Figure 12.3 The grammar for \mathcal{L}_0 , with example phrases for each rule.

(12.1) [S [NP [Pro I]] [VP [V prefer] [NP [Det a] [Nom [N morning] [Nom [N flight]]]]]]]

grammatical
ungrammaticalgenerative
grammar

A CFG like that of \mathcal{L}_0 defines a formal language. We saw in Chapter 2 that a formal language is a set of strings. Sentences (strings of words) that can be derived by a grammar are in the formal language defined by that grammar, and are called **grammatical** sentences. Sentences that cannot be derived by a given formal grammar are not in the language defined by that grammar and are referred to as **ungrammatical**. This hard line between “in” and “out” characterizes all formal languages but is only a very simplified model of how natural languages really work. This is because determining whether a given sentence is part of a given natural language (say, English) often depends on the context. In linguistics, the use of formal languages to model natural languages is called **generative grammar** since the language is defined by the set of possible sentences “generated” by the grammar.

12.2.1 Formal Definition of Context-Free Grammar

We conclude this section with a quick, formal description of a context-free grammar and the language it generates. A context-free grammar G is defined by four parameters: N, Σ, R, S (technically this is a “4-tuple”).

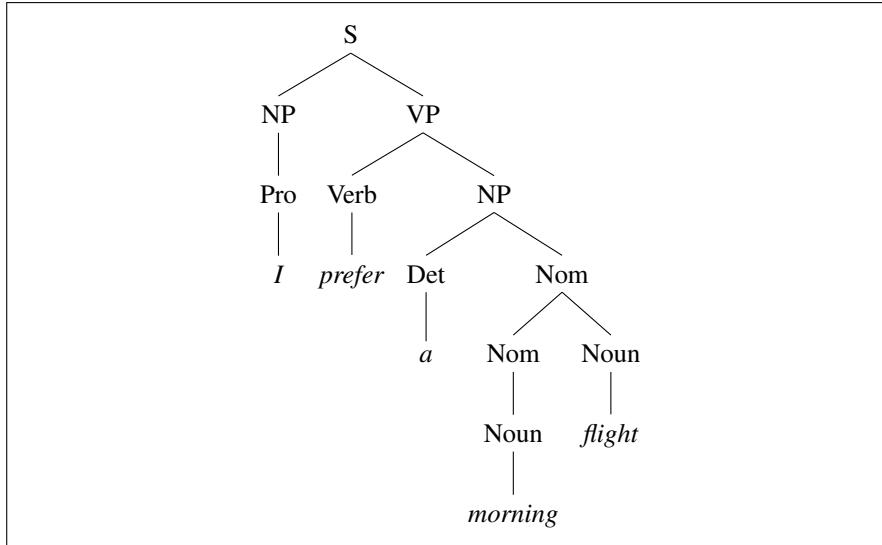


Figure 12.4 The parse tree for “I prefer a morning flight” according to grammar \mathcal{L}_0 .

N	a set of non-terminal symbols (or variables)
Σ	a set of terminal symbols (disjoint from N)
R	a set of rules or productions, each of the form $A \rightarrow \beta$,
	where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$
S	a designated start symbol and a member of N

For the remainder of the book we adhere to the following conventions when discussing the formal properties of context-free grammars (as opposed to explaining particular facts about English or other languages).

Capital letters like A , B , and S	Non-terminals
S	The start symbol
Lower-case Greek letters like α , β , and γ	Strings drawn from $(\Sigma \cup N)^*$
Lower-case Roman letters like u , v , and w	Strings of terminals

A language is defined through the concept of derivation. One string derives another one if it can be rewritten as the second one by some series of rule applications. More formally, following Hopcroft and Ullman (1979),

directly derives if $A \rightarrow \beta$ is a production of R and α and γ are any strings in the set $(\Sigma \cup N)^*$, then we say that $\alpha A \gamma$ **directly derives** $\alpha \beta \gamma$, or $\alpha A \gamma \Rightarrow \alpha \beta \gamma$.

Derivation is then a generalization of direct derivation:

Let $\alpha_1, \alpha_2, \dots, \alpha_m$ be strings in $(\Sigma \cup N)^*$, $m \geq 1$, such that

$$\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \dots, \alpha_{m-1} \Rightarrow \alpha_m$$

derives We say that α_1 **derives** α_m , or $\alpha_1 \xrightarrow{*} \alpha_m$.

We can then formally define the language \mathcal{L}_G generated by a grammar G as the set of strings composed of terminal symbols that can be derived from the designated

start symbol S .

$$\mathcal{L}_G = \{w \mid w \text{ is in } \Sigma^* \text{ and } S \xrightarrow{*} w\}$$

**syntactic
parsing**

The problem of mapping from a string of words to its parse tree is called **syntactic parsing**; we define algorithms for parsing in Chapter 13.

12.3 Some Grammar Rules for English

In this section, we introduce a few more aspects of the phrase structure of English; for consistency we will continue to focus on sentences from the ATIS domain. Because of space limitations, our discussion is necessarily limited to highlights. Readers are strongly advised to consult a good reference grammar of English, such as [Huddleston and Pullum \(2002\)](#).

12.3.1 Sentence-Level Constructions

In the small grammar \mathcal{L}_0 , we provided only one sentence-level construction for declarative sentences like *I prefer a morning flight*. Among the large number of constructions for English sentences, four are particularly common and important: declaratives, imperatives, yes-no questions, and wh-questions.

declarative

Sentences with **declarative** structure have a subject noun phrase followed by a verb phrase, like “I prefer a morning flight”. Sentences with this structure have a great number of different uses that we follow up on in Chapter 26. Here are a number of examples from the ATIS domain:

I want a flight from Ontario to Chicago
 The flight should be eleven a.m. tomorrow
 The return flight should leave at around seven p.m.

imperative

Sentences with **imperative** structure often begin with a verb phrase and have no subject. They are called imperative because they are almost always used for commands and suggestions; in the ATIS domain they are commands to the system.

Show the lowest fare
 Give me Sunday’s flights arriving in Las Vegas from New York City
 List all flights between five and seven p.m.

We can model this sentence structure with another rule for the expansion of S :

$$S \rightarrow VP$$

yes-no question

Sentences with **yes-no question** structure are often (though not always) used to ask questions; they begin with an auxiliary verb, followed by a subject NP , followed by a VP . Here are some examples. Note that the third example is not a question at all but a request; Chapter 26 discusses the uses of these question forms to perform different **pragmatic** functions such as asking, requesting, or suggesting.

Do any of these flights have stops?
 Does American’s flight eighteen twenty five serve dinner?
 Can you give me the same information for United?

Here’s the rule:

$$S \rightarrow Aux \ NP \ VP$$

wh-phrase The most complex sentence-level structures we examine here are the various **wh-** structures. These are so named because one of their constituents is a **wh-phrase**, that is, one that includes a **wh-word** (*who*, *whose*, *when*, *where*, *what*, *which*, *how*, *why*). These may be broadly grouped into two classes of sentence-level structures. The **wh-subject-question** structure is identical to the declarative structure, except that the first noun phrase contains some wh-word.

wh-word

What airlines fly from Burbank to Denver?

Which flights depart Burbank after noon and arrive in Denver by six p.m?

Whose flights serve breakfast?

Here is a rule. Exercise 12.7 discusses rules for the constituents that make up the *Wh-NP*.

$$S \rightarrow Wh-NP VP$$

wh-non-subject-question

In the **wh-non-subject-question** structure, the wh-phrase is not the subject of the sentence, and so the sentence includes another subject. In these types of sentences the auxiliary appears before the subject *NP*, just as in the yes-no question structures. Here is an example followed by a sample rule:

What flights do you have from Burbank to Tacoma Washington?

$$S \rightarrow Wh-NP Aux NP VP$$

long-distance dependencies

Constructions like the **wh-non-subject-question** contain what are called **long-distance dependencies** because the *Wh-NP what flights* is far away from the predicate that it is semantically related to, the main verb *have* in the *VP*. In some models of parsing and understanding compatible with the grammar rule above, long-distance dependencies like the relation between *flights* and *have* are thought of as a semantic relation. In such models, the job of figuring out that *flights* is the argument of *have* is done during semantic interpretation. In other models of parsing, the relationship between *flights* and *have* is considered to be a syntactic relation, and the grammar is modified to insert a small marker called a **trace** or **empty category** after the verb. We return to such empty-category models when we introduce the Penn Treebank on page 217.

12.3.2 Clauses and Sentences

Before we move on, we should clarify the status of the *S* rules in the grammars we just described. *S* rules are intended to account for entire sentences that stand alone as fundamental units of discourse. However, *S* can also occur on the right-hand side of grammar rules and hence can be embedded within larger sentences. Clearly then, there's more to being an *S* than just standing alone as a unit of discourse.

clause

What differentiates sentence constructions (i.e., the *S* rules) from the rest of the grammar is the notion that they are in some sense *complete*. In this way they correspond to the notion of a **clause**, which traditional grammars often describe as forming a complete thought. One way of making this notion of “complete thought” more precise is to say an *S* is a node of the parse tree below which the main verb of the *S* has all of its **arguments**. We define verbal arguments later, but for now let's just see an illustration from the tree for *I prefer a morning flight* in Fig. 12.4 on page 208. The verb *prefer* has two arguments: the subject *I* and the object *a morning flight*. One of the arguments appears below the *VP* node, but the other one, the subject *NP*, appears only below the *S* node.

12.3.3 The Noun Phrase

Our \mathcal{L}_0 grammar introduced three of the most frequent types of noun phrases that occur in English: pronouns, proper nouns and the $NP \rightarrow Det\ Nominal$ construction. The central focus of this section is on the last type since that is where the bulk of the syntactic complexity resides. These noun phrases consist of a head, the central noun in the noun phrase, along with various modifiers that can occur before or after the head noun. Let's take a close look at the various parts.

The Determiner

Noun phrases can begin with simple lexical determiners, as in the following examples:

a stop	the flights	this flight
those flights	any flights	some flights

The role of the determiner in English noun phrases can also be filled by more complex expressions, as follows:

United's flight
 United's pilot's union
 Denver's mayor's mother's canceled flight

In these examples, the role of the determiner is filled by a possessive expression consisting of a noun phrase followed by an '*'s*' as a possessive marker, as in the following rule.

$$Det \rightarrow NP\ 's$$

The fact that this rule is recursive (since an NP can start with a Det) helps us model the last two examples above, in which a sequence of possessive expressions serves as a determiner.

Under some circumstances determiners are optional in English. For example, determiners may be omitted if the noun they modify is plural:

(12.2) Show me *flights* from San Francisco to Denver on weekdays

As we saw in Chapter 8, **mass nouns** also don't require determination. Recall that mass nouns often (not always) involve something that is treated like a substance (including e.g., *water* and *snow*), don't take the indefinite article "*a*", and don't tend to pluralize. Many abstract nouns are mass nouns (*music*, *homework*). Mass nouns in the ATIS domain include *breakfast*, *lunch*, and *dinner*:

(12.3) Does this flight serve dinner?

The Nominal

The nominal construction follows the determiner and contains any pre- and post-head noun modifiers. As indicated in grammar \mathcal{L}_0 , in its simplest form a nominal can consist of a single noun.

$$Nominal \rightarrow Noun$$

As we'll see, this rule also provides the basis for the bottom of various recursive rules used to capture more complex nominal constructions.

Before the Head Noun

Cardinal numbers

**ordinal numbers
quantifiers**

A number of different kinds of word classes can appear before the head noun (the “postdeterminers”) in a nominal. These include **cardinal numbers**, **ordinal numbers**, **quantifiers**, and adjectives. Examples of cardinal numbers:

two friends one stop

Ordinal numbers include *first*, *second*, *third*, and so on, but also words like *next*, *last*, *past*, *other*, and *another*:

the first one	the next day	the second leg
the last flight	the other American flight	

Some quantifiers (*many*, *(a) few*, *several*) occur only with plural count nouns:

many fares

Adjectives occur after quantifiers but before nouns.

a <i>first-class</i> fare	a <i>non-stop</i> flight
the <i>longest</i> layover	the <i>earliest</i> lunch flight

adjective phrase

Adjectives can also be grouped into a phrase called an **adjective phrase** or AP. APs can have an adverb before the adjective (see Chapter 8 for definitions of adjectives and adverbs):

the *least expensive* fare

After the Head Noun

A head noun can be followed by **postmodifiers**. Three kinds of nominal postmodifiers are common in English:

prepositional phrases	all flights <i>from Cleveland</i>
non-finite clauses	any flights <i>arriving after eleven a.m.</i>
relative clauses	a flight <i>that serves breakfast</i>

They are especially common in the ATIS corpus since they are used to mark the origin and destination of flights.

Here are some examples of prepositional phrase postmodifiers, with brackets inserted to show the boundaries of each PP; note that two or more PPs can be strung together within a single NP:

all flights [*from Cleveland*] [*to Newark*]
 arrival [*in San Jose*] [*before seven p.m.*]
 a reservation [*on flight six oh six*] [*from Tampa*] [*to Montreal*]

Here's a new nominal rule to account for postnominal PPs:

$$\text{Nominal} \rightarrow \text{Nominal PP}$$

non-finite

The three most common kinds of **non-finite** postmodifiers are the gerundive (-*ing*), *-ed*, and infinitive forms.

gerundive

Gerundive postmodifiers are so called because they consist of a verb phrase that begins with the gerundive (-*ing*) form of the verb. Here are some examples:

any of those [*leaving on Thursday*]
 any flights [*arriving after eleven a.m.*]
 flights [*arriving within thirty minutes of each other*]

We can define the *Nominals* with gerundive modifiers as follows, making use of a new non-terminal *GerundVP*:

$$\text{Nominal} \rightarrow \text{Nominal GerundVP}$$

We can make rules for *GerundVP* constituents by duplicating all of our VP productions, substituting *GerundV* for *V*:

$$\begin{aligned}\text{GerundVP} &\rightarrow \text{GerundV NP} \\ &| \quad \text{GerundV PP} | \text{GerundV} | \text{GerundV NP PP}\end{aligned}$$

GerundV can then be defined as

$$\text{GerundV} \rightarrow \text{being} | \text{arriving} | \text{leaving} | \dots$$

The phrases in italics below are examples of the two other common kinds of non-finite clauses, infinitives and -ed forms:

the last flight *to arrive in Boston*
 I need to have dinner *served*
 Which is the aircraft *used by this flight?*

relative pronoun

A postnominal relative clause (more correctly a **restrictive relative clause**), is a clause that often begins with a **relative pronoun** (*that* and *who* are the most common). The relative pronoun functions as the subject of the embedded verb in the following examples:

a flight *that serves breakfast*
 flights *that leave in the morning*
 the one *that leaves at ten thirty five*

We might add rules like the following to deal with these:

$$\begin{aligned}\text{Nominal} &\rightarrow \text{Nominal RelClause} \\ \text{RelClause} &\rightarrow (\text{who} | \text{that}) \text{ VP}\end{aligned}$$

The relative pronoun may also function as the object of the embedded verb, as in the following example; we leave for the reader the exercise of writing grammar rules for more complex relative clauses of this kind.

the earliest American Airlines flight *that I can get*

Various postnominal modifiers can be combined, as the following examples show:

a flight [*from Phoenix to Detroit*] [*leaving Monday evening*]
 evening flights [*from Nashville to Houston*] [*that serve dinner*]
 a friend [*living in Denver*] [*that would like to visit me in DC*]

Before the Noun Phrase

predeterminers

Word classes that modify and appear before *NPs* are called **predeterminers**. Many of these have to do with number or amount; a common predeterminer is *all*:

all the flights all flights all non-stop flights

The example noun phrase given in Fig. 12.5 illustrates some of the complexity that arises when these rules are combined.

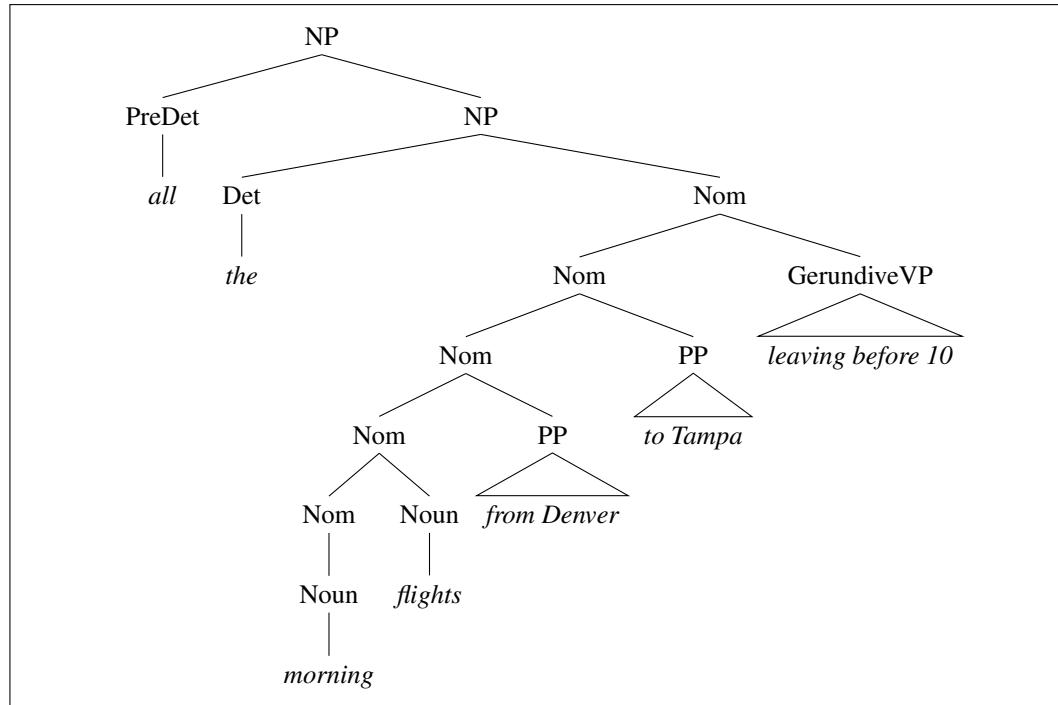


Figure 12.5 A parse tree for “all the morning flights from Denver to Tampa leaving before 10”.

12.3.4 The Verb Phrase

The verb phrase consists of the verb and a number of other constituents. In the simple rules we have built so far, these other constituents include *NPs* and *PPs* and combinations of the two:

$$\begin{aligned}
 VP &\rightarrow \text{Verb} \quad \text{disappear} \\
 VP &\rightarrow \text{Verb } NP \quad \text{prefer a morning flight} \\
 VP &\rightarrow \text{Verb } NP \text{ } PP \quad \text{leave Boston in the morning} \\
 VP &\rightarrow \text{Verb } PP \quad \text{leaving on Thursday}
 \end{aligned}$$

sentential
complements

Verb phrases can be significantly more complicated than this. Many other kinds of constituents, such as an entire embedded sentence, can follow the verb. These are called **sentential complements**:

You [VP [V said [_S you had a two hundred sixty-six dollar fare]]]
 [VP [V Tell] [NP me] [_S how to get from the airport in Philadelphia to downtown]]
 I [VP [V think [_S I would like to take the nine thirty flight]]]

Here's a rule for these:

$$VP \rightarrow \text{Verb } S$$

Similarly, another potential constituent of the *VP* is another *VP*. This is often the case for verbs like *want*, *would like*, *try*, *intend*, *need*:

I want [VP to fly from Milwaukee to Orlando]
 Hi, I want [VP to arrange three flights]

Frame	Verb	Example
\emptyset	eat, sleep	I ate
NP	prefer, find, leave	Find [NP the flight from Pittsburgh to Boston]
$NP\ NP$	show, give	Show [NP me] [NP airlines with flights from Pittsburgh]
$PP_{from}\ PP_{to}$	fly, travel	I would like to fly [PP from Boston] [PP to Philadelphia]
$NP\ PP_{with}$	help, load	Can you help [NP me] [PP with a flight]
VP_{to}	prefer, want, need	I would prefer [VP_{to} to go by United Airlines]
VP_{brst}	can, would, might	I can [VP_{brst} go from Boston]
S	mean	Does this mean [S AA has a hub in Boston]

Figure 12.6 Subcategorization frames for a set of example verbs.

While a verb phrase can have many possible kinds of constituents, not every verb is compatible with every verb phrase. For example, the verb *want* can be used either with an *NP* complement (*I want a flight ...*) or with an infinitive *VP* complement (*I want to fly to ...*). By contrast, a verb like *find* cannot take this sort of *VP* complement (**I found to fly to Dallas*).

This idea that verbs are compatible with different kinds of complements is a very old one; traditional grammar distinguishes between **transitive** verbs like *find*, which take a direct object *NP* (*I found a flight*), and **intransitive** verbs like *disappear*, which do not (**I disappeared a flight*).

transitive
intransitive
subcategorize
Subcategorizes for
complements
Subcategorization frame

Where traditional grammars **subcategorize** verbs into these two categories (transitive and intransitive), modern grammars distinguish as many as 100 subcategories. We say that a verb like *find* **subcategorizes for** an *NP*, and a verb like *want* subcategorizes for either an *NP* or a non-finite *VP*. We also call these constituents the **complements** of the verb (hence our use of the term **sentential complement** above). So we say that *want* can take a *VP* complement. These possible sets of complements are called the **subcategorization frame** for the verb. Another way of talking about the relation between the verb and these other constituents is to think of the verb as a logical predicate and the constituents as logical arguments of the predicate. So we can think of such predicate-argument relations as FIND(I, A FLIGHT) or WANT(I, TO FLY). We talk more about this view of verbs and arguments in Chapter 16 when we talk about predicate calculus representations of verb semantics. Subcategorization frames for a set of example verbs are given in Fig. 12.6.

We can capture the association between verbs and their complements by making separate subtypes of the class Verb (e.g., *Verb-with-NP-complement*, *Verb-with-Inf-VP-complement*, *Verb-with-S-complement*, and so on):

Verb-with-NP-complement → *find* | *leave* | *repeat* | ...

Verb-with-S-complement → *think* | *believe* | *say* | ...

Verb-with-Inf-VP-complement → *want* | *try* | *need* | ...

Each *VP* rule could then be modified to require the appropriate verb subtype:

VP → *Verb-with-no-complement* disappear

VP → *Verb-with-NP-comp* *NP* prefer a morning flight

VP → *Verb-with-S-comp* *S* said there were two flights

A problem with this approach is the significant increase in the number of rules and the associated loss of generality.

12.3.5 Coordination

conjunctions The major phrase types discussed here can be conjoined with **conjunctions** like *and*, *or*, and *but* to form larger constructions of the same type. For example, a **coordinate** noun phrase can consist of two other noun phrases separated by a conjunction:

Please repeat [*NP* [*NP* the flights] *and* [*NP* the costs]]
 I need to know [*NP* [*NP* the aircraft] *and* [*NP* the flight number]]

Here's a rule that allows these structures:

$$NP \rightarrow NP \text{ and } NP$$

Note that the ability to form coordinate phrases through conjunctions is often used as a test for constituency. Consider the following examples, which differ from the ones given above in that they lack the second determiner.

Please repeat the [*Nom* [*Nom* flights] *and* [*Nom* costs]]
 I need to know the [*Nom* [*Nom* aircraft] *and* [*Nom* flight number]]

The fact that these phrases can be conjoined is evidence for the presence of the underlying *Nominal* constituent we have been making use of. Here's a rule for this:

$$Nominal \rightarrow Nominal \text{ and } Nominal$$

The following examples illustrate conjunctions involving *VPs* and *Ss*.

What flights do you have [*VP* [*VP* leaving Denver] *and* [*VP* arriving in San Francisco]]
 [*s* [*s* I'm interested in a flight from Dallas to Washington] *and* [*s* I'm also interested in going to Baltimore]]

The rules for *VP* and *S* conjunctions mirror the *NP* one given above.

$$VP \rightarrow VP \text{ and } VP$$

$$S \rightarrow S \text{ and } S$$

Since all the major phrase types can be conjoined in this fashion, it is also possible to represent this conjunction fact more generally; a number of grammar formalisms such as GPSG ((Gazdar et al., 1985)) do this using **metarules** such as the following:

$$X \rightarrow X \text{ and } X$$

This metarule simply states that any non-terminal can be conjoined with the same non-terminal to yield a constituent of the same type. Of course, the variable *X* must be designated as a variable that stands for any non-terminal rather than a non-terminal itself.

12.4 Treebanks

treebank Sufficiently robust grammars consisting of context-free grammar rules can be used to assign a parse tree to any sentence. This means that it is possible to build a corpus where every sentence in the collection is paired with a corresponding parse tree. Such a syntactically annotated corpus is called a **treebank**. Treebanks play

an important role in parsing, as we discuss in Chapter 13, as well as in linguistic investigations of syntactic phenomena.

Penn Treebank

A wide variety of treebanks have been created, generally through the use of parsers (of the sort described in the next few chapters) to automatically parse each sentence, followed by the use of humans (linguists) to hand-correct the parses. The **Penn Treebank** project (whose POS tagset we introduced in Chapter 8) has produced treebanks from the Brown, Switchboard, ATIS, and *Wall Street Journal* corpora of English, as well as treebanks in Arabic and Chinese. A number of treebanks use the dependency representation we will introduce in Chapter 15, including many that are part of the **Universal Dependencies** project (Nivre et al., 2016b).

12.4.1 Example: The Penn Treebank Project

Figure 12.7 shows sentences from the Brown and ATIS portions of the Penn Treebank.¹ Note the formatting differences for the part-of-speech tags; such small differences are common and must be dealt with in processing treebanks. The Penn Treebank part-of-speech tagset was defined in Chapter 8. The use of LISP-style parenthesized notation for trees is extremely common and resembles the bracketed notation we saw earlier in (12.1). For those who are not familiar with it we show a standard node-and-line tree representation in Fig. 12.8.

<pre>((S (NP-SBJ (DT That) (JJ cold) (, ,) (JJ empty) (NN sky)) (VP (VBD was) (ADJP-PRD (JJ full) (PP (IN of) (NP (NN fire) (CC and) (NN light))))) (. .)))</pre> <p style="text-align: center;">(a)</p>	<pre>((S (NP-SBJ The/DT flight/NN) (VP should/MD (VP arrive/VB (PP-TMP at/IN (NP eleven/CD a.m/RB)) (NP-TMP tomorrow/NN)))))</pre> <p style="text-align: center;">(b)</p>
--	---

Figure 12.7 Parsed sentences from the LDC Treebank3 version of the Brown (a) and ATIS (b) corpora.

traces
syntactic
movement

Figure 12.9 shows a tree from the *Wall Street Journal*. This tree shows another feature of the Penn Treebanks: the use of **traces** (-NONE- nodes) to mark long-distance dependencies or **syntactic movement**. For example, quotations often follow a quotative verb like *say*. But in this example, the quotation “We would have to wait until we have collected on those assets” precedes the words *he said*. An empty *S* containing only the node -NONE- marks the position after *said* where the quotation sentence often occurs. This empty node is marked (in Treebanks II and III) with the index 2, as is the quotation *S* at the beginning of the sentence. Such co-indexing may make it easier for some parsers to recover the fact that this fronted or topicalized quotation is the complement of the verb *said*. A similar -NONE- node

¹ The Penn Treebank project released treebanks in multiple languages and in various stages; for example, there were Treebank I (Marcus et al., 1993), Treebank II (Marcus et al., 1994), and Treebank III releases of English treebanks. We use Treebank III for our examples.



Figure 12.8 The tree corresponding to the Brown corpus sentence in the previous figure.

marks the fact that there is no syntactic subject right before the verb *to wait*; instead, the subject is the earlier *NP We*. Again, they are both co-indexed with the index 1.



Figure 12.9 A sentence from the *Wall Street Journal* portion of the LDC Penn Treebank. Note the use of the empty -NONE- nodes.

The Penn Treebank II and Treebank III releases added further information to make it easier to recover the relationships between predicates and arguments. Cer-

Grammar	Lexicon
$S \rightarrow NP VP.$	$PRP \rightarrow we he$
$S \rightarrow NP VP$	$DT \rightarrow the that those$
$S \rightarrow "S", NP VP.$	$JJ \rightarrow cold empty full$
$S \rightarrow -NONE-$	$NN \rightarrow sky fire light flight tomorrow$
$NP \rightarrow DT NN$	$NNS \rightarrow assets$
$NP \rightarrow DT NNS$	$CC \rightarrow and$
$NP \rightarrow NN CC NN$	$IN \rightarrow of at until on$
$NP \rightarrow CD RB$	$CD \rightarrow eleven$
$NP \rightarrow DT JJ, JJ NN$	$RB \rightarrow a.m.$
$NP \rightarrow PRP$	$VB \rightarrow arrive have wait$
$NP \rightarrow -NONE-$	$VBD \rightarrow was said$
$VP \rightarrow MD VP$	$VBP \rightarrow have$
$VP \rightarrow VBD ADJP$	$VBN \rightarrow collected$
$VP \rightarrow VBD S$	$MD \rightarrow should would$
$VP \rightarrow VBN PP$	$TO \rightarrow to$
$VP \rightarrow VB S$	
$VP \rightarrow VB SBAR$	
$VP \rightarrow VBP VP$	
$VP \rightarrow VBN PP$	
$VP \rightarrow TO VP$	
$SBAR \rightarrow IN S$	
$ADJP \rightarrow JJ PP$	
$PP \rightarrow IN NP$	

Figure 12.10 A sample of the CFG grammar rules and lexical entries that would be extracted from the three treebank sentences in Fig. 12.7 and Fig. 12.9.

tain phrases were marked with tags indicating the grammatical function of the phrase (as surface subject, logical topic, cleft, non-VP predicates) its presence in particular text categories (headlines, titles), and its semantic function (temporal phrases, locations) (Marcus et al. 1994, Bies et al. 1995). Figure 12.9 shows examples of the -SBJ (surface subject) and -TMP (temporal phrase) tags. Figure 12.8 shows in addition the -PRD tag, which is used for predicates that are not VPs (the one in Fig. 12.8 is an ADJP). We'll return to the topic of grammatical function when we consider dependency grammars and parsing in Chapter 15.

12.4.2 Treebanks as Grammars

The sentences in a treebank implicitly constitute a grammar of the language represented by the corpus being annotated. For example, from the three parsed sentences in Fig. 12.7 and Fig. 12.9, we can extract each of the CFG rules in them. For simplicity, let's strip off the rule suffixes (-SBJ and so on). The resulting grammar is shown in Fig. 12.10.

The grammar used to parse the Penn Treebank is relatively flat, resulting in very many and very long rules. For example, among the approximately 4,500 different rules for expanding VPs are separate rules for PP sequences of any length and every possible arrangement of verb arguments:

```

VP → VBD PP
VP → VBD PP PP
VP → VBD PP PP PP
VP → VBD PP PP PP PP
VP → VB ADVP PP
VP → VB PP ADVP
VP → ADVP VB PP

```

as well as even longer rules, such as

$\text{VP} \rightarrow \text{VBP PP PP PP PP PP ADVP PP}$

which comes from the *VP* marked in italics:

This mostly happens because we *go from football in the fall to lifting in the winter to football again in the spring.*

Some of the many thousands of *NP* rules include

```

NP → DT JJ NN
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ ‘‘ SBAR ’’ NNS

```

The last two of those rules, for example, come from the following two noun phrases:

[DT The] [JJ state-owned] [JJ industrial] [VBG holding] [NN company] [NNP Instituto]
 [NNP Nacional] [FW de] [NNP Industria]
 [NP Shearson’s] [JJ easy-to-film], [JJ black-and-white] “[SBAR Where We Stand]”
 [NNS commercials]

Viewed as a large grammar in this way, the Penn Treebank III *Wall Street Journal* corpus, which contains about 1 million words, also has about 1 million non-lexical rule tokens, consisting of about 17,500 distinct rule types.

Various facts about the treebank grammars, such as their large numbers of flat rules, pose problems for probabilistic parsing algorithms. For this reason, it is common to make various modifications to a grammar extracted from a treebank. We discuss these further in Chapter 14.

12.4.3 Heads and Head Finding

We suggested informally earlier that syntactic constituents could be associated with a lexical **head**; *N* is the head of an *NP*, *V* is the head of a *VP*. This idea of a head for each constituent dates back to Bloomfield (1914). It is central to constituent-based grammar formalisms such as Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994), as well as the dependency-based approaches to grammar we’ll discuss in Chapter 15. Heads and head-dependent relations have also come to play a central role in computational linguistics with their use in probabilistic parsing (Chapter 14) and in dependency parsing (Chapter 15).

In one simple model of lexical heads, each context-free rule is associated with a head (Charniak 1997, Collins 1999). The head is the word in the phrase that is grammatically the most important. Heads are passed up the parse tree; thus, each non-terminal in a parse tree is annotated with a single word, which is its lexical head.

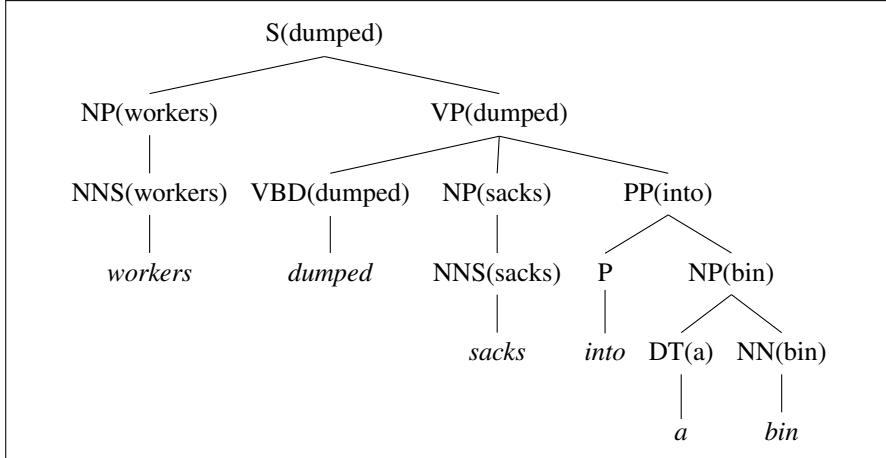


Figure 12.11 A lexicalized tree from [Collins \(1999\)](#).

Figure 12.11 shows an example of such a tree from [Collins \(1999\)](#), in which each non-terminal is annotated with its head.

For the generation of such a tree, each CFG rule must be augmented to identify one right-side constituent to be the head daughter. The headword for a node is then set to the headword of its head daughter. Choosing these head daughters is simple for textbook examples (*NN* is the head of *NP*) but is complicated and indeed controversial for most phrases. (Should the complementizer *to* or the verb be the head of an infinite verb-phrase?) Modern linguistic theories of syntax generally include a component that defines heads (see, e.g., [\(Pollard and Sag, 1994\)](#)).

An alternative approach to finding a head is used in most practical computational systems. Instead of specifying head rules in the grammar itself, heads are identified dynamically in the context of trees for specific sentences. In other words, once a sentence is parsed, the resulting tree is walked to decorate each node with the appropriate head. Most current systems rely on a simple set of handwritten rules, such as a practical one for Penn Treebank grammars given in [Collins \(1999\)](#) but developed originally by [Magerman \(1995\)](#). For example, the rule for finding the head of an *NP* is as follows ([Collins, 1999, p. 238](#)):

- If the last word is tagged POS, return last-word.
- Else search from right to left for the first child which is an *NN*, *NNP*, *NNPS*, *NX*, *POS*, or *JJR*.
- Else search from left to right for the first child which is an *NP*.
- Else search from right to left for the first child which is a \$, *ADJP*, or *PRN*.
- Else search from right to left for the first child which is a *CD*.
- Else search from right to left for the first child which is a *JJ*, *JJS*, *RB* or *QP*.
- Else return the last word

Selected other rules from this set are shown in Fig. 12.12. For example, for *VP* rules of the form $VP \rightarrow Y_1 \dots Y_n$, the algorithm would start from the left of $Y_1 \dots Y_n$ looking for the first Y_i of type *TO*; if no *TO*s are found, it would search for the first Y_i of type *VBD*; if no *VBD*s are found, it would search for a *VBN*, and so on. See [Collins \(1999\)](#) for more details.

Parent	Direction	Priority List
ADJP	Left	NNS QP NN \$ ADVP JJ VBN VBG ADJP JJR NP JJS DT FW RBR RBS SBAR RB
ADVP	Right	RB RBR RBS FW ADVP TO CD JJR JJ IN NP JJS NN
PRN	Left	
PRT	Right	RP
QP	Left	\$ IN NNS NN JJ RB DT CD NCD QP JJR JJS
S	Left	TO IN VP S SBAR ADJP UCP NP
SBAR	Left	WHNP WHPP WHADVP WHADJP IN DT S SQ SINV SBAR FRAG
VP	Left	TO VBD VBN MD VBZ VB VBG VBP VP ADJP NN NNS NP

Figure 12.12 Selected head rules from [Collins \(1999\)](#). The set of head rules is often called a **head percolation table**.

12.5 Grammar Equivalence and Normal Form

A formal language is defined as a (possibly infinite) set of strings of words. This suggests that we could ask if two grammars are equivalent by asking if they generate the same set of strings. In fact, it is possible to have two distinct context-free grammars generate the same language.

We usually distinguish two kinds of grammar equivalence: **weak equivalence** and **strong equivalence**. Two grammars are strongly equivalent if they generate the same set of strings *and* if they assign the same phrase structure to each sentence (allowing merely for renaming of the non-terminal symbols). Two grammars are weakly equivalent if they generate the same set of strings but do not assign the same phrase structure to each sentence.

normal form

Chomsky normal form

binary branching

It is sometimes useful to have a **normal form** for grammars, in which each of the productions takes a particular form. For example, a context-free grammar is in **Chomsky normal form** (CNF) ([Chomsky, 1963](#)) if it is ϵ -free and if in addition each production is either of the form $A \rightarrow B C$ or $A \rightarrow a$. That is, the right-hand side of each rule either has two non-terminal symbols or one terminal symbol. Chomsky normal form grammars are **binary branching**, that is they have binary trees (down to the prelexical nodes). We make use of this binary branching property in the CKY parsing algorithm in Chapter 13.

Any context-free grammar can be converted into a weakly equivalent Chomsky normal form grammar. For example, a rule of the form

$$A \rightarrow B C D$$

can be converted into the following two CNF rules (Exercise 12.8 asks the reader to formulate the complete algorithm):

$$\begin{aligned} A &\rightarrow B X \\ X &\rightarrow C D \end{aligned}$$

Sometimes using binary branching can actually produce smaller grammars. For example, the sentences that might be characterized as

$$VP \rightarrow VBD \ NP \ PP^*$$

are represented in the Penn Treebank by this series of rules:

$$VP \rightarrow VBD \ NP \ PP$$

$$VP \rightarrow VBD \ NP \ PP \ PP$$

$\text{VP} \rightarrow \text{VBD } \text{NP } \text{PP } \text{PP } \text{PP}$
 $\text{VP} \rightarrow \text{VBD } \text{NP } \text{PP } \text{PP } \text{PP } \text{PP}$
 ...

but could also be generated by the following two-rule grammar:

$\text{VP} \rightarrow \text{VBD } \text{NP } \text{PP}$
 $\text{VP} \rightarrow \text{VP } \text{PP}$

Chomsky-adjunction

The generation of a symbol A with a potentially infinite sequence of symbols B with a rule of the form $A \rightarrow A B$ is known as **Chomsky-adjunction**.

12.6 Lexicalized Grammars

The approach to grammar presented thus far emphasizes phrase-structure rules while minimizing the role of the lexicon. However, as we saw in the discussions of agreement, subcategorization, and long-distance dependencies, this approach leads to solutions that are cumbersome at best, yielding grammars that are redundant, hard to manage, and brittle. To overcome these issues, numerous alternative approaches have been developed that all share the common theme of making better use of the lexicon. Among the more computationally relevant approaches are Lexical-Functional Grammar (LFG) (Bresnan, 1982), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994), Tree-Adjoining Grammar (TAG) (Joshi, 1985), and Combinatory Categorial Grammar (CCG). These approaches differ with respect to how *lexicalized* they are — the degree to which they rely on the lexicon as opposed to phrase structure rules to capture facts about the language.

The following section provides an introduction to CCG, a heavily lexicalized approach motivated by both syntactic and semantic considerations, which we will return to in Chapter 16. Chapter 15 discusses dependency grammars, an approach that eliminates phrase-structure rules entirely.

12.6.1 Combinatory Categorial Grammar

categorial grammar
combinatory categorial grammar

In this section, we provide an overview of **categorial grammar** (Ajdukiewicz 1935, Bar-Hillel 1953), an early lexicalized grammar model, as well as an important modern extension, **combinatory categorial grammar**, or CCG (Steedman 1996, Steedman 1989, Steedman 2000).

The categorial approach consists of three major elements: a set of categories, a lexicon that associates words with categories, and a set of rules that govern how categories combine in context.

Categories

Categories are either atomic elements or single-argument functions that return a category as a value when provided with a desired category as argument. More formally, we can define \mathcal{C} , a set of categories for a grammar as follows:

- $\mathcal{A} \subseteq \mathcal{C}$, where \mathcal{A} is a given set of atomic elements
- $(X/Y), (X \setminus Y) \in \mathcal{C}$, if $X, Y \in \mathcal{C}$

The slash notation shown here is used to define the functions in the grammar. It specifies the type of the expected argument, the direction it is expected be found, and the type of the result. Thus, (X/Y) is a function that seeks a constituent of type

Y to its right and returns a value of X ; $(X \setminus Y)$ is the same except it seeks its argument to the left.

The set of atomic categories is typically very small and includes familiar elements such as sentences and noun phrases. Functional categories include verb phrases and complex noun phrases among others.

The Lexicon

The lexicon in a categorial approach consists of assignments of categories to words. These assignments can either be to atomic or functional categories, and due to lexical ambiguity words can be assigned to multiple categories. Consider the following sample lexical entries.

<i>flight</i> :	<i>N</i>
<i>Miami</i> :	<i>NP</i>
<i>cancel</i> :	$(S \setminus NP)/NP$

Nouns and proper nouns like *flight* and *Miami* are assigned to atomic categories, reflecting their typical role as arguments to functions. On the other hand, a transitive verb like *cancel* is assigned the category $(S \setminus NP)/NP$: a function that seeks an *NP* on its right and returns as its value a function with the type $(S \setminus NP)$. This function can, in turn, combine with an *NP* on the left, yielding an *S* as the result. This captures the kind of subcategorization information discussed in Section 12.3.4, however here the information has a rich, computationally useful, internal structure.

Ditransitive verbs like *give*, which expect two arguments after the verb, would have the category $((S \setminus NP)/NP)/NP$: a function that combines with an *NP* on its right to yield yet another function corresponding to the transitive verb $(S \setminus NP)/NP$ category such as the one given above for *cancel*.

Rules

The rules of a categorial grammar specify how functions and their arguments combine. The following two rule templates constitute the basis for all categorial grammars.

$$X/Y \ Y \Rightarrow X \tag{12.4}$$

$$Y \ X \setminus Y \Rightarrow X \tag{12.5}$$

The first rule applies a function to its argument on the right, while the second looks to the left for its argument. We'll refer to the first as **forward function application**, and the second as **backward function application**. The result of applying either of these rules is the category specified as the value of the function being applied.

Given these rules and a simple lexicon, let's consider an analysis of the sentence *United serves Miami*. Assume that *serves* is a transitive verb with the category $(S \setminus NP)/NP$ and that *United* and *Miami* are both simple *NPs*. Using both forward and backward function application, the derivation would proceed as follows:

$$\begin{array}{c} \frac{\begin{array}{ccc} \text{United} & \text{serves} & \text{Miami} \\ \hline \text{NP} & \frac{(S \setminus NP)/NP}{\text{NP}} & \text{NP} \end{array}}{\frac{\text{S} \setminus \text{NP}}{\text{S}}} \end{array}$$

Categorial grammar derivations are illustrated growing down from the words, rule applications are illustrated with a horizontal line that spans the elements involved, with the type of the operation indicated at the right end of the line. In this example, there are two function applications: one forward function application indicated by the $>$ that applies the verb *serves* to the *NP* on its right, and one backward function application indicated by the $<$ that applies the result of the first to the *NP United* on its left.

With the addition of another rule, the categorial approach provides a straightforward way to implement the coordination metarule described earlier on page 216. Recall that English permits the coordination of two constituents of the same type, resulting in a new constituent of the same type. The following rule provides the mechanism to handle such examples.

$$X \text{ CONJ } X \Rightarrow X \quad (12.6)$$

This rule states that when two constituents of the same category are separated by a constituent of type *CONJ* they can be combined into a single larger constituent of the same type. The following derivation illustrates the use of this rule.



Here the two *S\NP* constituents are combined via the conjunction operator $\langle \Phi \rangle$ to form a larger constituent of the same type, which can then be combined with the subject *NP* via backward function application.

These examples illustrate the lexical nature of the categorial grammar approach. The grammatical facts about a language are largely encoded in the lexicon, while the rules of the grammar are boiled down to a set of three rules. Unfortunately, the basic categorial approach does not give us any more expressive power than we had with traditional CFG rules; it just moves information from the grammar to the lexicon. To move beyond these limitations CCG includes operations that operate over functions.

The first pair of operators permit us to **compose** adjacent functions.

$$X/Y \text{ } Y/Z \Rightarrow X/Z \quad (12.7)$$

$$Y\backslash Z \text{ } X\backslash Y \Rightarrow X\backslash Z \quad (12.8)$$

forward composition

The first rule, called **forward composition**, can be applied to adjacent constituents where the first is a function seeking an argument of type *Y* to its right, and the second is a function that provides *Y* as a result. This rule allows us to compose these two functions into a single one with the type of the first constituent and the argument of the second. Although the notation is a little awkward, the second rule, **backward composition** is the same, except that we're looking to the left instead of to the right for the relevant arguments. Both kinds of composition are signalled by a **B** in CCG diagrams, accompanied by a $<$ or $>$ to indicate the direction.

backward composition

type raising

The next operator is **type raising**. Type raising elevates simple categories to the status of functions. More specifically, type raising takes a category and converts it to function that seeks as an argument a function that takes the original category

as its argument. The following schema show two versions of type raising: one for arguments to the right, and one for the left.

$$X \Rightarrow T/(T \setminus X) \quad (12.9)$$

$$X \Rightarrow T \setminus (T/X) \quad (12.10)$$

The category T in these rules can correspond to any of the atomic or functional categories already present in the grammar.

A particularly useful example of type raising transforms a simple NP argument in subject position to a function that can compose with a following VP . To see how this works, let's revisit our earlier example of *United serves Miami*. Instead of classifying *United* as an NP which can serve as an argument to the function attached to *serves*, we can use type raising to reinvent it as a function in its own right as follows.

$$NP \Rightarrow S/(S \setminus NP)$$

Combining this type-raised constituent with the forward composition rule (12.7) permits the following alternative to our previous derivation.

$$\begin{array}{c} \begin{array}{ccc} \textit{United} & \textit{serves} & \textit{Miami} \\ \hline \text{NP} & (S \setminus NP)/NP & \text{NP} \end{array} \\ \xrightarrow{\text{S}/(S \setminus NP)} \\ \xrightarrow{\text{S}/\text{NP}} \\ \xrightarrow{\text{S}} \end{array}$$

By type raising *United* to $S/(S \setminus NP)$, we can compose it with the transitive verb *serves* to yield the (S/NP) function needed to complete the derivation.

There are several interesting things to note about this derivation. First, is it provides a left-to-right, word-by-word derivation that more closely mirrors the way humans process language. This makes CCG a particularly apt framework for psycholinguistic studies. Second, this derivation involves the use of an intermediate unit of analysis, *United serves*, that does not correspond to a traditional constituent in English. This ability to make use of such non-constituent elements provides CCG with the ability to handle the coordination of phrases that are not proper constituents, as in the following example.

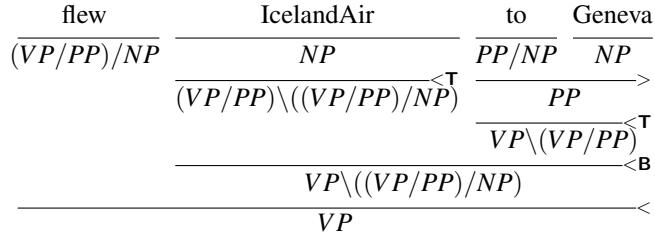
(12.11) We flew IcelandAir to Geneva and SwissAir to London.

Here, the segments that are being coordinated are *IcelandAir to Geneva* and *SwissAir to London*, phrases that would not normally be considered constituents, as can be seen in the following standard derivation for the verb phrase *flew IcelandAir to Geneva*.

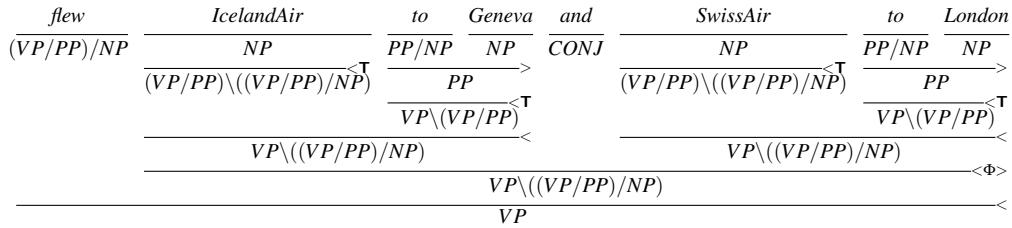
$$\begin{array}{ccccccccc} \textit{flew} & \textit{IcelandAir} & \textit{to} & \textit{Geneva} & & & & & \\ \hline (\text{VP}/\text{PP})/\text{NP} & \text{NP} & \text{PP}/\text{NP} & \text{NP} & & & & & \\ \xrightarrow{\text{VP}/\text{PP}} & & \xrightarrow{\text{PP}} & & & & & & \\ \xrightarrow{\text{VP}} & & & & & & & & \end{array}$$

In this derivation, there is no single constituent that corresponds to *IcelandAir to Geneva*, and hence no opportunity to make use of the $\langle\Phi\rangle$ operator. Note that complex CCG categories can get a little cumbersome, so we'll use VP as a shorthand for $(S \setminus NP)$ in this and the following derivations.

The following alternative derivation provides the required element through the use of both backward type raising (12.10) and backward function composition (12.8).



Applying the same analysis to *SwissAir to London* satisfies the requirements for the $\langle\Phi\rangle$ operator, yielding the following derivation for our original example (12.11).

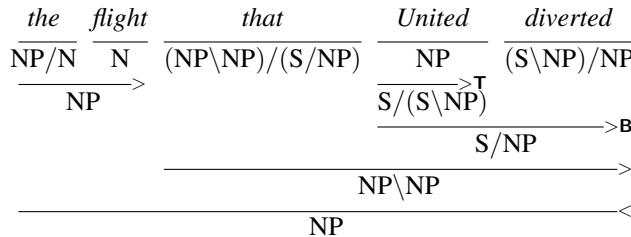


Finally, let's examine how these advanced operators can be used to handle **long-distance dependencies** (also referred to as syntactic movement or extraction). As mentioned in Section 12.3.1, long-distance dependencies arise from many English constructions including wh-questions, relative clauses, and topicalization. What these constructions have in common is a constituent that appears somewhere distant from its usual, or expected, location. Consider the following relative clause as an example.

the flight that United diverted

Here, *divert* is a transitive verb that expects two *NP* arguments, a subject *NP* to its left and a direct object *NP* to its right; its category is therefore $(S\backslash NP)/NP$. However, in this example the direct object *the flight* has been “moved” to the beginning of the clause, while the subject *United* remains in its normal position. What is needed is a way to incorporate the subject argument, while dealing with the fact that *the flight* is not in its expected location.

The following derivation accomplishes this, again through the combined use of type raising and function composition.



As we saw with our earlier examples, the first step of this derivation is type raising *United* to the category $S/(S\backslash NP)$ allowing it to combine with *diverted* via forward composition. The result of this composition is S/NP which preserves the fact that we are still looking for an *NP* to fill the missing direct object. The second critical piece is the lexical category assigned to the word *that*: $(NP\backslash NP)/(S/NP)$. This function seeks a verb phrase missing an argument to its right, and transforms it into an *NP* seeking a missing element to its left, precisely where we find *the flight*.

CCGBank

As with phrase-structure approaches, treebanks play an important role in CCG-based approaches to parsing. CCGBank ([Hockenmaier and Steedman, 2007](#)) is the largest and most widely used CCG treebank. It was created by automatically translating phrase-structure trees from the Penn Treebank via a rule-based approach. The method produced successful translations of over 99% of the trees in the Penn Treebank resulting in 48,934 sentences paired with CCG derivations. It also provides a lexicon of 44,000 words with over 1200 categories. Chapter 14 will discuss how these resources can be used to train CCG parsers.

12.7 Summary

This chapter has introduced a number of fundamental concepts in syntax through the use of **context-free grammars**.

- In many languages, groups of consecutive words act as a group or a **constituent**, which can be modeled by **context-free grammars** (which are also known as **phrase-structure grammars**).
- A context-free grammar consists of a set of **rules** or **productions**, expressed over a set of **non-terminal** symbols and a set of **terminal** symbols. Formally, a particular **context-free language** is the set of strings that can be **derived** from a particular **context-free grammar**.
- A **generative grammar** is a traditional name in linguistics for a formal language that is used to model the grammar of a natural language.
- There are many sentence-level grammatical constructions in English; **declarative**, **imperative**, **yes-no question**, and **wh-question** are four common types; these can be modeled with context-free rules.
- An English **noun phrase** can have **determiners**, **numbers**, **quantifiers**, and **adjective phrases** preceding the **head noun**, which can be followed by a number of **postmodifiers**; **gerundive VPs**, **infinitives VPs**, and **past participial VPs** are common possibilities.
- **Subjects** in English **agree** with the main verb in person and number.
- Verbs can be **subcategorized** by the types of **complements** they expect. Simple subcategories are **transitive** and **intransitive**; most grammars include many more categories than these.
- **Treebanks** of parsed sentences exist for many genres of English and for many languages. Treebanks can be searched with tree-search tools.
- Any context-free grammar can be converted to **Chomsky normal form**, in which the right-hand side of each rule has either two non-terminals or a single terminal.
- Lexicalized grammars place more emphasis on the structure of the lexicon, lessening the burden on pure phrase-structure rules.
- Combinatorial categorial grammar (CCG) is an important computationally relevant lexicalized approach.

Bibliographical and Historical Notes

[The origin of the idea of phrasal constituency, cited in [Percival \(1976\)](#)]:

*...den sprachlichen Ausdruck für die willkürliche
Gliederung einer Gesamtvorstellung in ihre*

in logische Beziehung zueinander gesetzten Bestandteile
[the linguistic expression for the arbitrary division of a total idea
into its constituent parts placed in logical relations to one another]

W. Wundt

According to [Percival \(1976\)](#), the idea of breaking up a sentence into a hierarchy of constituents appeared in the *Völkerpsychologie* of the groundbreaking psychologist Wilhelm Wundt ([Wundt, 1900](#)). Wundt's idea of constituency was taken up into linguistics by Leonard Bloomfield in his early book *An Introduction to the Study of Language* ([Bloomfield, 1914](#)). By the time of his later book, *Language* ([Bloomfield, 1933](#)), what was then called “immediate-constituent analysis” was a well-established method of syntactic study in the United States. By contrast, traditional European grammar, dating from the Classical period, defined relations between *words* rather than constituents, and European syntacticians retained this emphasis on such **dependency** grammars, the subject of Chapter 15.

American Structuralism saw a number of specific definitions of the immediate constituent, couched in terms of their search for a “discovery procedure”: a methodological algorithm for describing the syntax of a language. In general, these attempt to capture the intuition that “The primary criterion of the immediate constituent is the degree in which combinations behave as simple units” ([Bazell, 1966, p. 284](#)). The most well known of the specific definitions is Harris' idea of distributional similarity to individual units, with the *substitutability* test. Essentially, the method proceeded by breaking up a construction into constituents by attempting to substitute simple structures for possible constituents—if a substitution of a simple form, say, *man*, was substitutable in a construction for a more complex set (like *intense young man*), then the form *intense young man* was probably a constituent. Harris's test was the beginning of the intuition that a constituent is a kind of equivalence class.

The first formalization of this idea of hierarchical constituency was the **phrase-structure grammar** defined in [Chomsky \(1956\)](#) and further expanded upon (and argued against) in [Chomsky \(1957\)](#) and [Chomsky \(1975\)](#). From this time on, most generative linguistic theories were based at least in part on context-free grammars or generalizations of them (such as Head-Driven Phrase Structure Grammar ([Pollard and Sag, 1994](#)), Lexical-Functional Grammar ([Bresnan, 1982](#)), Government and Binding ([Chomsky, 1981](#)), and Construction Grammar ([Kay and Fillmore, 1999](#)), *inter alia*); many of these theories used schematic context-free templates known as **X-bar schemata**, which also relied on the notion of syntactic head.

**X-bar
schemata**

Shortly after Chomsky's initial work, the context-free grammar was reinvented by [Backus \(1959\)](#) and independently by [Naur et al. \(1960\)](#) in their descriptions of the ALGOL programming language; [Backus \(1996\)](#) noted that he was influenced by the productions of Emil Post and that Naur's work was independent of his (Backus') own. (Recall the discussion on page ?? of multiple invention in science.) After this early work, a great number of computational models of natural language processing were based on context-free grammars because of the early development of efficient algorithms to parse these grammars (see Chapter 13).

As we have already noted, grammars based on context-free rules are not ubiquitous. Various classes of extensions to CFGs are designed specifically to handle long-distance dependencies. We noted earlier that some grammars treat long-distance-dependent items as being related semantically but not syntactically; the surface syntax does not represent the long-distance link ([Kay and Fillmore 1999](#), [Culicover and Jackendoff 2005](#)). But there are alternatives.

One extended formalism is **Tree Adjoining Grammar** (TAG) ([Joshi, 1985](#)). The primary TAG data structure is the tree, rather than the rule. Trees come in two kinds: **initial trees** and **auxiliary trees**. Initial trees might, for example, represent simple sentential structures, and auxiliary trees add recursion into a tree. Trees are combined by two operations called **substitution** and **adjunction**. The adjunction operation handles long-distance dependencies. See [Joshi \(1985\)](#) for more details. An extension of Tree Adjoining Grammar, called Lexicalized Tree Adjoining Grammars is discussed in Chapter 14. Tree Adjoining Grammar is a member of the family of **mildly context-sensitive languages**.

We mentioned on page 217 another way of handling long-distance dependencies, based on the use of empty categories and co-indexing. The Penn Treebank uses this model, which draws (in various Treebank corpora) from the Extended Standard Theory and Minimalism ([Radford, 1997](#)).

Readers interested in the grammar of English should get one of the three large reference grammars of English: [Huddleston and Pullum \(2002\)](#), [Biber et al. \(1999\)](#), and [Quirk et al. \(1985\)](#). Another useful reference is [McCawley \(1998\)](#).

There are many good introductory textbooks on syntax from different perspectives. [Sag et al. \(2003\)](#) is an introduction to syntax from a **generative** perspective, focusing on the use of phrase-structure rules, unification, and the type hierarchy in Head-Driven Phrase Structure Grammar. [Van Valin, Jr. and La Polla \(1997\)](#) is an introduction from a **functional** perspective, focusing on cross-linguistic data and on the functional motivation for syntactic structures.

Exercises

12.1 Draw tree structures for the following ATIS phrases:

1. Dallas
2. from Denver
3. after five p.m.
4. arriving in Washington
5. early flights
6. all redeye flights
7. on Thursday
8. a one-way fare
9. any delays in Denver

12.2 Draw tree structures for the following ATIS sentences:

1. Does American Airlines have a flight between five a.m. and six a.m.?
2. I would like to fly on American Airlines.
3. Please repeat that.
4. Does American 487 have a first-class section?
5. I need to fly between Philadelphia and Atlanta.
6. What is the fare from Atlanta to Denver?

7. Is there an American Airlines flight from Philadelphia to Dallas?

- 12.3** Assume a grammar that has many *VP* rules for different subcategorizations, as expressed in Section 12.3.4, and differently subcategorized verb rules like *Verb-with-NP-complement*. How would the rule for postnominal relative clauses (12.4) need to be modified if we wanted to deal properly with examples like *the earliest flight that you have*? Recall that in such examples the pronoun *that* is the object of the verb *get*. Your rules should allow this noun phrase but should correctly rule out the ungrammatical *S *I get*.
- 12.4** Does your solution to the previous problem correctly model the NP *the earliest flight that I can get*? How about *the earliest flight that I think my mother wants me to book for her*? Hint: this phenomenon is called **long-distance dependency**.
- 12.5** Write rules expressing the verbal subcategory of English auxiliaries; for example, you might have a rule *verb-with-bare-stem-VP-complement → can*.
- 12.6** NPs like *Fortune's office* or *my uncle's marks* are called **possessive** or **genitive** noun phrases. We can model possessive noun phrases by treating the sub-NP like *Fortune's* or *my uncle's* as a determiner of the following head noun. Write grammar rules for English possessives. You may treat 's as if it were a separate word (i.e., as if there were always a space before 's).
- 12.7** Page 210 discussed the need for a *Wh-NP* constituent. The simplest *Wh-NP* is one of the *Wh-pronouns* (*who*, *whom*, *whose*, *which*). The *Wh*-words *what* and *which* can be determiners: *which four will you have?*, *what credit do you have with the Duke?* Write rules for the different types of *Wh-NPs*.
- 12.8** Write an algorithm for converting an arbitrary context-free grammar into Chomsky normal form.

possessive
genitive

CHAPTER

13

Constituency Parsing

*One morning I shot an elephant in my pajamas.
How he got into my pajamas I don't know.*

Groucho Marx, *Animal Crackers*, 1930

Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it. This chapter focuses on the structures assigned by context-free grammars of the kind described in Chapter 12. Since they are based on a purely declarative formalism, context-free grammars don't specify *how* the parse tree for a given sentence should be computed. We therefore need to specify algorithms that employ these grammars to efficiently produce correct trees.

Parse trees are directly useful in applications such as **grammar checking** in word-processing systems: a sentence that cannot be parsed may have grammatical errors (or at least be hard to read). More typically, however, parse trees serve as an important intermediate stage of representation for **semantic analysis** (as we show in Chapter 17) and thus play an important role in applications like **question answering** and **information extraction**. For example, to answer the question

What books were written by British women authors before 1800?

we'll need to know that the subject of the sentence was *what books* and that the by-adjunct was *British women authors* to help us figure out that the user wants a list of books (and not a list of authors).

Before presenting any algorithms, we begin by discussing how the ambiguity arises again in this context and the problems it presents. The section that follows then presents the Cocke-Kasami-Younger (CKY) algorithm (Kasami 1965, Younger 1967), the standard dynamic programming approach to syntactic parsing. Recall that we've already seen applications of dynamic programming algorithms in the Minimum-Edit-Distance and Viterbi algorithms of earlier chapters. Finally, we discuss **partial parsing methods**, for use in situations in which a superficial syntactic analysis of an input may be sufficient.

13.1 Ambiguity

structural
ambiguity

Ambiguity is perhaps the most serious problem faced by syntactic parsers. Chapter 8 introduced the notions of **part-of-speech ambiguity** and **part-of-speech disambiguation**. Here, we introduce a new kind of ambiguity, called **structural ambiguity**, which arises from many commonly used rules in phrase-structure grammars. To illustrate the issues associated with structural ambiguity, we'll make use of a new toy grammar \mathcal{L}_1 , shown in Figure 13.1, which consists of the \mathcal{L}_0 grammar from the last chapter augmented with a few additional rules.

Structural ambiguity occurs when the grammar can assign more than one parse to a sentence. Groucho Marx's well-known line as Captain Spaulding in *Animal*

Grammar	Lexicon
$S \rightarrow NP VP$	$Det \rightarrow that this the a$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book flight meal money$
$S \rightarrow VP$	$Verb \rightarrow book include prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I she me$
$NP \rightarrow Proper-Noun$	$Proper-Noun \rightarrow Houston NWA$
$NP \rightarrow Det Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from to on near through$
$Nominal \rightarrow Nominal Noun$	
$Nominal \rightarrow Nominal PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	
$VP \rightarrow Verb NP PP$	
$VP \rightarrow Verb PP$	
$VP \rightarrow VP PP$	
$PP \rightarrow Preposition NP$	

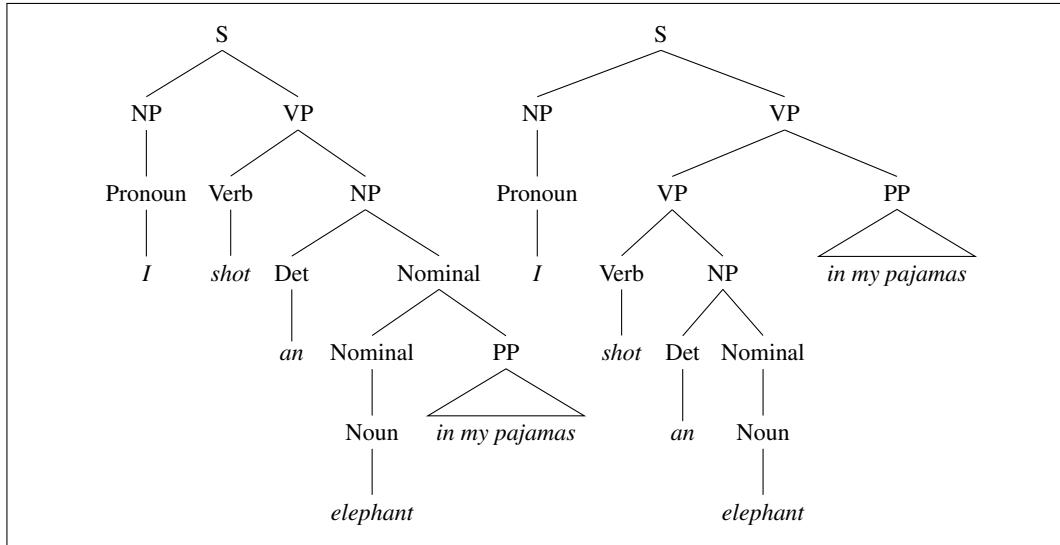
Figure 13.1 The \mathcal{L}_1 miniature English grammar and lexicon.

Figure 13.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

Crackers is ambiguous because the phrase *in my pajamas* can be part of the *NP* headed by *elephant* or a part of the verb phrase headed by *shot*. Figure 13.2 illustrates these two analyses of Marx's line using rules from \mathcal{L}_1 .

Structural ambiguity, appropriately enough, comes in many forms. Two common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**.

attachment ambiguity

A sentence has an **attachment ambiguity** if a particular constituent can be attached to the parse tree at more than one place. The Groucho Marx sentence is an example of *PP*-attachment ambiguity. Various kinds of adverbial phrases are also subject to this kind of ambiguity. For instance, in the following example the gerundive-*VP* *flying to Paris* can be part of a gerundive sentence whose subject is *the Eiffel Tower* or it can be an adjunct modifying the *VP* headed by *saw*:

(13.1) We saw the Eiffel Tower flying to Paris.

coordination ambiguity

In **coordination ambiguity** different sets of phrases can be conjoined by a conjunction like *and*. For example, the phrase *old men and women* can be bracketed as *[old [men and women]]*, referring to *old men* and *old women*, or as *[old men] and [women]*, in which case it is only the men who are old.

These ambiguities combine in complex ways in real sentences. A program that summarized the news, for example, would need to be able to parse sentences like the following from the Brown corpus:

- (13.2) President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

This sentence has a number of ambiguities, although since they are semantically unreasonable, it requires a careful reading to see them. The last noun phrase could be parsed *[nationwide [television and radio]]* or *[[nationwide television] and radio]*. The direct object of *pushed aside* should be *other White House business* but could also be the bizarre phrase *[other White House business to devote all his time and attention to working]* (i.e., a structure like *Kennedy affirmed [his intention to propose a new budget to address the deficit]*). Then the phrase *on the Berlin crisis address he will deliver tomorrow night to the American people* could be an adjunct modifying the verb *pushed*. A PP like *over nationwide television and radio* could be attached to any of the higher VPs or NPs (e.g., it could modify *people* or *night*).

syntactic disambiguation

The fact that there are many grammatically correct but semantically unreasonable parses for naturally occurring sentences is an irksome problem that affects all parsers. Ultimately, most natural language processing systems need to be able to choose a single correct parse from the multitude of possible parses through a process of **syntactic disambiguation**. Effective disambiguation algorithms require statistical, semantic, and contextual knowledge sources that vary in how well they can be integrated into parsing algorithms.

Fortunately, the CKY algorithm presented in the next section is designed to efficiently handle structural ambiguities of the kind we've been discussing. And as we'll see in Chapter 14, there are straightforward ways to integrate statistical techniques into the basic CKY framework to produce highly accurate parsers.

13.2 CKY Parsing: A Dynamic Programming Approach

The previous section introduced some of the problems associated with ambiguous grammars. Fortunately, **dynamic programming** provides a powerful framework for addressing these problems, just as it did with the Minimum Edit Distance, Viterbi, and Forward algorithms. Recall that dynamic programming approaches systematically fill in tables of solutions to sub-problems. When complete, the tables contain the solution to all the sub-problems needed to solve the problem as a whole. In the case of syntactic parsing, these sub-problems represent parse trees for all the constituents detected in the input.

The dynamic programming advantage arises from the context-free nature of our grammar rules — once a constituent has been discovered in a segment of the input we can record its presence and make it available for use in any subsequent derivation that might require it. This provides both time and storage efficiencies since subtrees can be looked up in a table, not reanalyzed. This section presents the Cocke-Kasami-

Younger (CKY) algorithm, the most widely used dynamic-programming based approach to parsing. Related approaches include the **Earley algorithm** (Earley, 1970) and **chart parsing** (Kaplan 1973, Kay 1982).

13.2.1 Conversion to Chomsky Normal Form

We begin our investigation of the CKY algorithm by examining the requirement that grammars used with it must be in Chomsky Normal Form (CNF). Recall from Chapter 12 that grammars in CNF are restricted to rules of the form $A \rightarrow BC$ or $A \rightarrow w$. That is, the right-hand side of each rule must expand either to two non-terminals or to a single terminal. Restricting a grammar to CNF does not lead to any loss in expressiveness, since any context-free grammar can be converted into a corresponding CNF grammar that accepts exactly the same set of strings as the original grammar.

Let's start with the process of converting a generic CFG into one represented in CNF. Assuming we're dealing with an ϵ -free grammar, there are three situations we need to address in any generic grammar: rules that mix terminals with non-terminals on the right-hand side, rules that have a single non-terminal on the right-hand side, and rules in which the length of the right-hand side is greater than 2.

The remedy for rules that mix terminals and non-terminals is to simply introduce a new dummy non-terminal that covers only the original terminal. For example, a rule for an infinitive verb phrase such as $INF-VP \rightarrow to VP$ would be replaced by the two rules $INF-VP \rightarrow TO VP$ and $TO \rightarrow to$.

Unit
productions

Rules with a single non-terminal on the right are called **unit productions**. We can eliminate unit productions by rewriting the right-hand side of the original rules with the right-hand side of all the non-unit production rules that they ultimately lead to. More formally, if $A \xrightarrow{*} B$ by a chain of one or more unit productions and $B \rightarrow \gamma$ is a non-unit production in our grammar, then we add $A \rightarrow \gamma$ for each such rule in the grammar and discard all the intervening unit productions. As we demonstrate with our toy grammar, this can lead to a substantial *flattening* of the grammar and a consequent promotion of terminals to fairly high levels in the resulting trees.

Rules with right-hand sides longer than 2 are normalized through the introduction of new non-terminals that spread the longer sequences over several new rules. Formally, if we have a rule like

$$A \rightarrow BC\gamma$$

we replace the leftmost pair of non-terminals with a new non-terminal and introduce a new production result in the following new rules:

$$\begin{aligned} A &\rightarrow X1\gamma \\ X1 &\rightarrow BC \end{aligned}$$

In the case of longer right-hand sides, we simply iterate this process until the offending rule has been replaced by rules of length 2. The choice of replacing the leftmost pair of non-terminals is purely arbitrary; any systematic scheme that results in binary rules would suffice.

In our current grammar, the rule $S \rightarrow Aux NP VP$ would be replaced by the two rules $S \rightarrow X1 VP$ and $X1 \rightarrow Aux NP$.

The entire conversion process can be summarized as follows:

1. Copy all conforming rules to the new grammar unchanged.

\mathcal{L}_1 Grammar	\mathcal{L}_1 in CNF
$S \rightarrow NP VP$	$S \rightarrow NP VP$
$S \rightarrow Aux NP VP$	$S \rightarrow X1 VP$
$S \rightarrow VP$	$X1 \rightarrow Aux NP$
	$S \rightarrow book include prefer$
	$S \rightarrow Verb NP$
	$S \rightarrow X2 PP$
	$S \rightarrow Verb PP$
	$S \rightarrow VP PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I she me$
$NP \rightarrow Proper-Noun$	$NP \rightarrow TWA Houston$
$NP \rightarrow Det Nominal$	$NP \rightarrow Det Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book flight meal money$
$Nominal \rightarrow Nominal Noun$	$Nominal \rightarrow Nominal Noun$
$Nominal \rightarrow Nominal PP$	$Nominal \rightarrow Nominal PP$
$VP \rightarrow Verb$	$VP \rightarrow book include prefer$
$VP \rightarrow Verb NP$	$VP \rightarrow Verb NP$
$VP \rightarrow Verb NP PP$	$VP \rightarrow X2 PP$
$VP \rightarrow Verb PP$	$X2 \rightarrow Verb NP$
$VP \rightarrow VP PP$	$VP \rightarrow Verb PP$
$PP \rightarrow Preposition NP$	$VP \rightarrow VP PP$
	$PP \rightarrow Preposition NP$

Figure 13.3 \mathcal{L}_1 Grammar and its conversion to CNF. Note that although they aren't shown here, all the original lexical entries from \mathcal{L}_1 carry over unchanged as well.

2. Convert terminals within rules to dummy non-terminals.
3. Convert unit productions.
4. Make all rules binary and add them to new grammar.

Figure 13.3 shows the results of applying this entire conversion procedure to the \mathcal{L}_1 grammar introduced earlier on page 233. Note that this figure doesn't show the original lexical rules; since these original lexical rules are already in CNF, they all carry over unchanged to the new grammar. Figure 13.3 does, however, show the various places where the process of eliminating unit productions has, in effect, created new lexical rules. For example, all the original verbs have been promoted to both VPs and to Ss in the converted grammar.

13.2.2 CKY Recognition

With our grammar now in CNF, each non-terminal node above the part-of-speech level in a parse tree will have exactly two daughters. A two-dimensional matrix can be used to encode the structure of an entire tree. For a sentence of length n , we will work with the upper-triangular portion of an $(n+1) \times (n+1)$ matrix. Each cell $[i, j]$ in this matrix contains the set of non-terminals that represent all the constituents that span positions i through j of the input. Since our indexing scheme begins with 0, it's natural to think of the indexes as pointing at the gaps between the input words (as in $_0 Book_1 that_2 flight_3$). It follows then that the cell that represents the entire input resides in position $[0, n]$ in the matrix.

Since each non-terminal entry in our table has two daughters in the parse, it follows that for each constituent represented by an entry $[i, j]$, there must be a position in the input, k , where it can be split into two parts such that $i < k < j$. Given such

a position k , the first constituent $[i, k]$ must lie to the left of entry $[i, j]$ somewhere along row i , and the second entry $[k, j]$ must lie beneath it, along column j .

To make this more concrete, consider the following example with its completed parse matrix, shown in Fig. 13.4.

(13.3) Book the flight through Houston.

The superdiagonal row in the matrix contains the parts of speech for each word in the input. The subsequent diagonals above that superdiagonal contain constituents that cover all the spans of increasing length in the input.

Book the flight through Houston				
S, VP, Verb Nominal, Noun [0,1]	S, VP,X2 [0,2]	S, VP,X2 [0,3]	S, VP,X2 [0,4]	S, VP,X2 [0,5]
Det [1,2]	NP [1,3]			NP [1,5]
	Nominal, Noun [2,3]			Nominal [2,5]
	Prep [3,4]		PP [3,5]	
			NP, Proper- Noun [4,5]	

Figure 13.4 Completed parse table for *Book the flight through Houston*.

Given this setup, CKY recognition consists of filling the parse table in the right way. To do this, we'll proceed in a bottom-up fashion so that at the point where we are filling any cell $[i, j]$, the cells containing the parts that could contribute to this entry (i.e., the cells to the left and the cells below) have already been filled. The algorithm given in Fig. 13.5 fills the upper-triangular matrix a column at a time working from left to right, with each column filled from bottom to top, as the right side of Fig. 13.4 illustrates. This scheme guarantees that at each point in time we have all the information we need (to the left, since all the columns to the left have already been filled, and below since we're filling bottom to top). It also mirrors on-line parsing since filling the columns from left to right corresponds to processing each word one at a time.

```

function CKY-PARSE(words, grammar) returns table
  for  $j \leftarrow$  from 1 to LENGTH(words) do
    for all { $A \mid A \rightarrow words[j] \in grammar$ } do
       $table[j-1, j] \leftarrow table[j-1, j] \cup A$ 
    for  $i \leftarrow$  from  $j-2$  downto 0 do
      for  $k \leftarrow i+1$  to  $j-1$  do
        for all { $A \mid A \rightarrow BC \in grammar$  and  $B \in table[i, k]$  and  $C \in table[k, j]$ } do
           $table[i, j] \leftarrow table[i, j] \cup A$ 

```

Figure 13.5 The CKY algorithm.

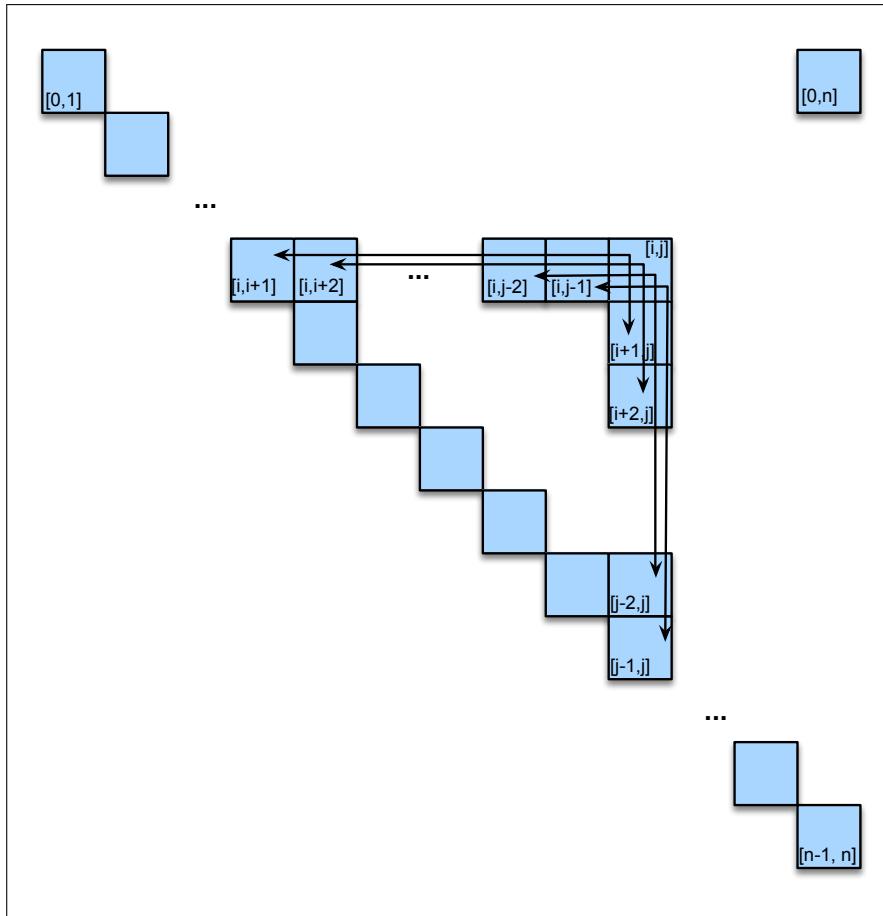


Figure 13.6 All the ways to fill the $[i, j]$ th cell in the CKY table.

The outermost loop of the algorithm given in Fig. 13.5 iterates over the columns, and the second loop iterates over the rows, from the bottom up. The purpose of the innermost loop is to range over all the places where a substring spanning i to j in the input might be split in two. As k ranges over the places where the string can be split, the pairs of cells we consider move, in lockstep, to the right along row i and down along column j . Figure 13.6 illustrates the general case of filling cell $[i, j]$. At each such split, the algorithm considers whether the contents of the two cells can be combined in a way that is sanctioned by a rule in the grammar. If such a rule exists, the non-terminal on its left-hand side is entered into the table.

Figure 13.7 shows how the five cells of column 5 of the table are filled after the word *Houston* is read. The arrows point out the two spans that are being used to add an entry to the table. Note that the action in cell $[0, 5]$ indicates the presence of three alternative parses for this input, one where the *PP* modifies the *flight*, one where it modifies the *booking*, and one that captures the second argument in the original $VP \rightarrow Verb\ NP\ PP$ rule, now captured indirectly with the $VP \rightarrow X2\ PP$ rule.

13.2.3 CKY Parsing

The algorithm given in Fig. 13.5 is a recognizer, not a parser; for it to succeed, it simply has to find an S in cell $[0, n]$. To turn it into a parser capable of returning all

Figure 13.7 Filling the cells of column 5 after reading the word *Houston*.

possible parses for a given input, we can make two simple changes to the algorithm: the first change is to augment the entries in the table so that each non-terminal is paired with pointers to the table entries from which it was derived (more or less as shown in Fig. 13.7), the second change is to permit multiple versions of the same non-terminal to be entered into the table (again as shown in Fig. 13.7). With these changes, the completed table contains all the possible parses for a given input. Returning an arbitrary single parse consists of choosing an S from cell $[0, n]$ and then recursively retrieving its component constituents from the table.

Of course, returning all the parses for a given input may incur considerable cost since an exponential number of parses may be associated with a given input. In such cases, returning all the parses will have an unavoidable exponential cost. Looking forward to Chapter 14, we can also think about retrieving the best parse for a given input by further augmenting the table to contain the probabilities of each entry. Retrieving the most probable parse consists of running a suitably modified version of the Viterbi algorithm from Chapter 8 over the completed parse table.

13.2.4 CKY in Practice

Finally, we should note that while the restriction to CNF does not pose a problem theoretically, it does pose some non-trivial problems in practice. Obviously, as things stand now, our parser isn't returning trees that are consistent with the grammar given to us by our friendly syntacticians. In addition to making our grammar developers unhappy, the conversion to CNF will complicate any syntax-driven approach to semantic analysis.

One approach to getting around these problems is to keep enough information around to transform our trees back to the original grammar as a post-processing step of the parse. This is trivial in the case of the transformation used for rules with length greater than 2. Simply deleting the new dummy non-terminals and promoting their daughters restores the original tree.

In the case of unit productions, it turns out to be more convenient to alter the basic CKY algorithm to handle them directly than it is to store the information needed to recover the correct trees. Exercise 13.3 asks you to make this change. Many of the probabilistic parsers presented in Chapter 14 use the CKY algorithm altered in just this manner. Another solution is to adopt a more complex dynamic programming solution that simply accepts arbitrary CFGs. The next section presents such an approach.

13.3 Partial Parsing

partial parse
shallow parse

Many language processing tasks do not require complex, complete parse trees for all inputs. For these tasks, a **partial parse**, or **shallow parse**, of input sentences may be sufficient. For example, information extraction systems generally do not extract *all* the possible information from a text: they simply identify and classify the segments in a text that are likely to contain valuable information. Similarly, information retrieval systems may index texts according to a subset of the constituents found in them.

There are many different approaches to partial parsing. Some make use of cascades of finite state transducers to produce tree-like representations. These approaches typically produce flatter trees than the ones we've been discussing in this

chapter and the previous one. This flatness arises from the fact that finite state transducer approaches generally defer decisions that may require semantic or contextual factors, such as prepositional phrase attachments, coordination ambiguities, and nominal compound analyses. Nevertheless, the intent is to produce parse trees that link all the major constituents in an input.

chunking

An alternative style of partial parsing is known as **chunking**. Chunking is the process of identifying and classifying the flat, non-overlapping segments of a sentence that constitute the basic non-recursive phrases corresponding to the major content-word parts-of-speech: noun phrases, verb phrases, adjective phrases, and prepositional phrases. The task of finding all the base noun phrases in a text is particularly common. Since chunked texts lack a hierarchical structure, a simple bracketing notation is sufficient to denote the location and the type of the chunks in a given example:

(13.4) [NP The morning flight] [PP from] [NP Denver] [VP has arrived.]

This bracketing notation makes clear the two fundamental tasks that are involved in chunking: segmenting (finding the non-overlapping extents of the chunks) and labeling (assigning the correct tag to the discovered chunks).

Some input words may not be part of any chunk, particularly in tasks like base *NP*:

(13.5) [NP The morning flight] from [NP Denver] has arrived.

What constitutes a syntactic base phrase depends on the application (and whether the phrases come from a treebank). Nevertheless, some standard guidelines are followed in most systems. First and foremost, base phrases of a given type do not recursively contain any constituents of the same type. Eliminating this kind of recursion leaves us with the problem of determining the boundaries of the non-recursive phrases. In most approaches, base phrases include the headword of the phrase, along with any pre-head material within the constituent, while crucially excluding any post-head material. Eliminating post-head modifiers obviates the need to resolve attachment ambiguities. This exclusion does lead to certain oddities, such as *PPs* and *VPs* often consisting solely of their heads. Thus, our earlier example *a flight from Indianapolis to Houston on NWA* is reduced to the following:

(13.6) [NP a flight] [PP from] [NP Indianapolis][PP to][NP Houston][PP on][NP NWA]

13.3.1 Machine Learning-Based Approaches to Chunking

State-of-the-art approaches to chunking use supervised machine learning to train a chunker by using annotated data as a training set and training any sequence labeler.

IOB

It's common to model chunking as **IOB** tagging. In IOB tagging we introduce a tag for the beginning (B) and inside (I) of each chunk type, and one for tokens outside (O) any chunk. The number of tags is thus $2n + 1$ tags, where n is the number of chunk types. IOB tagging can represent exactly the same information as the bracketed notation. The following example shows the bracketing notation of (13.4) on page 241 reframed as a tagging task:

(13.7) *The morning flight from Denver has arrived*
 B_NP I_NP I_NP B_PP B_NP B_VP I_VP

The same sentence with only the base-NPs tagged illustrates the role of the O tags.



Figure 13.8 A sequence model for chunking. The chunker slides a context window over the sentence, classifying words as it proceeds. At this point, the classifier is attempting to label *flight*, using features like words, embeddings, part-of-speech tags and previously assigned chunk tags.

(13.8) *The morning flight from Denver has arrived.*

B_NP I_NP I_NP O B_NP O O

There is no explicit encoding of the end of a chunk in IOB tagging; the end of any chunk is implicit in any transition from an I or B to a B or O tag. This encoding reflects the notion that when sequentially labeling words, it is generally easier (at least in English) to detect the beginning of a new chunk than it is to know when a chunk has ended.

Since annotation efforts are expensive and time consuming, chunkers usually rely on existing treebanks like the Penn Treebank (Chapter 12), extracting syntactic phrases from the full parse constituents of a sentence, finding the appropriate heads and then including the material to the left of the head, ignoring the text to the right. This is somewhat error-prone since it relies on the accuracy of the head-finding rules described in Chapter 12.

Given a training set, any sequence model can be used. Figure 13.8 shows an illustration of a simple feature-based model, using features like the words and parts-of-speech within a 2 word window, and the chunk tags of the preceding inputs in the window. In training, each training vector would consist of the values of 13 features; the two words to the left of the decision point, their parts-of-speech and chunk tags, the word to be tagged along with its part-of-speech, the two words that follow along with their parts-of speech, and the correct chunk tag, in this case, I_NP. During classification, the classifier is given the same vector without the answer and assigns the most appropriate tag from its tagset. Viterbi decoding is commonly used.

13.3.2 Chunking-System Evaluations

As with the evaluation of part-of-speech taggers, the evaluation of chunkers proceeds by comparing chunker output with gold-standard answers provided by human annotators. However, unlike part-of-speech tagging, word-by-word accuracy measures are not appropriate. Instead, chunkers are evaluated according to the notions of

precision, recall, and the **F-measure** that we saw in text classification in Chapter 4, repeated here as a quick refresher.

precision

Precision measures the percentage of system-provided chunks that were correct. Correct here means that both the boundaries of the chunk and the chunk's label are correct. Precision is therefore defined as

$$\text{Precision:} = \frac{\text{Number of correct chunks given by system}}{\text{Total number of chunks given by system}}$$

recall

Recall measures the percentage of chunks actually present in the input that were correctly identified by the system. Recall is defined as

$$\text{Recall:} = \frac{\text{Number of correct chunks given by system}}{\text{Total number of actual chunks in the text}}$$

F-measure

The **F-measure** (van Rijsbergen, 1975) provides a way to combine these two measures into a single metric:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

The β parameter differentially weights the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is sometimes called $F_{\beta=1}$ or just F_1 :

$$F_1 = \frac{2PR}{P+R} \quad (13.9)$$

13.4 Summary

The two major ideas introduced in this chapter are those of **parsing** and **partial parsing**. Here's a summary of the main points we covered about these ideas:

- **Structural ambiguity** is a significant problem for parsers. Common sources of structural ambiguity include **PP-attachment**, **coordination ambiguity**, and **noun-phrase bracketing ambiguity**.
- **Dynamic programming** parsing algorithms, such as **CKY**, use a table of partial parses to efficiently parse ambiguous sentences.
- **CKY** restricts the form of the grammar to Chomsky normal form (CNF).
- Many practical problems, including **information extraction** problems, can be solved without full parsing.
- **Partial parsing** and **chunking** are methods for identifying shallow syntactic constituents in a text.
- State-of-the-art methods for partial parsing use **supervised machine learning** techniques.

Bibliographical and Historical Notes

Writing about the history of compilers, Knuth notes:

In this field there has been an unusual amount of parallel discovery of the same technique by people working independently.

Well, perhaps not unusual, since multiple discovery is the norm in science (see page ??). But there has certainly been enough parallel publication that this history errs on the side of succinctness in giving only a characteristic early mention of each algorithm; the interested reader should see [Aho and Ullman \(1972\)](#).

WFST

Bottom-up parsing seems to have been first described by [Yngve \(1955\)](#), who gave a breadth-first, bottom-up parsing algorithm as part of an illustration of a machine translation procedure. Top-down approaches to parsing and translation were described (presumably independently) by at least [Glennie \(1960\)](#), [Irons \(1961\)](#), and [Kuno and Oettinger \(1963\)](#). Dynamic programming parsing, once again, has a history of independent discovery. According to Martin Kay (personal communication), a dynamic programming parser containing the roots of the CKY algorithm was first implemented by John Cocke in 1960. Later work extended and formalized the algorithm, as well as proving its time complexity ([Kay 1967](#), [Younger 1967](#), [Kasami 1965](#)). The related **well-formed substring table (WFST)** seems to have been independently proposed by [Kuno \(1965\)](#) as a data structure that stores the results of all previous computations in the course of the parse. Based on a generalization of Cocke's work, a similar data structure had been independently described in [Kay \(1967\)](#) (and [Kay 1973](#)). The top-down application of dynamic programming to parsing was described in Earley's Ph.D. dissertation ([Earley 1968](#), [Earley 1970](#)). [Sheil \(1976\)](#) showed the equivalence of the WFST and the Earley algorithm. [Norvig \(1991\)](#) shows that the efficiency offered by dynamic programming can be captured in any language with a *memoization* function (such as in LISP) simply by wrapping the *memoization* operation around a simple top-down parser.

While parsing via cascades of finite-state automata had been common in the early history of parsing ([Harris, 1962](#)), the focus shifted to full CFG parsing quite soon afterward. [Church \(1980\)](#) argued for a return to finite-state grammars as a processing model for natural language understanding; other early finite-state parsing models include [Ejehed \(1988\)](#). [Abney \(1991\)](#) argued for the important practical role of shallow parsing.

The classic reference for parsing algorithms is [Aho and Ullman \(1972\)](#); although the focus of that book is on computer languages, most of the algorithms have been applied to natural language. A good programming languages textbook such as [Aho et al. \(1986\)](#) is also useful.

Exercises

- 13.1** Implement the algorithm to convert arbitrary context-free grammars to CNF. Apply your program to the \mathcal{L}_1 grammar.
- 13.2** Implement the CKY algorithm and test it with your converted \mathcal{L}_1 grammar.
- 13.3** Rewrite the CKY algorithm given in Fig. 13.5 on page 237 so that it can accept grammars that contain unit productions.
- 13.4** Discuss the relative advantages and disadvantages of partial versus full parsing.

- 13.5** Discuss how to augment a parser to deal with input that may be incorrect, for example, containing spelling errors or mistakes arising from automatic speech recognition.

CHAPTER

14

Statistical Constituency Parsing

The characters in Damon Runyon’s short stories are willing to bet “on any proposition whatever”, as Runyon says about Sky Masterson in *The Idyll of Miss Sarah Brown*, from the probability of getting aces back-to-back to the odds against a man being able to throw a peanut from second base to home plate. There is a moral here for language processing: with enough knowledge we can figure the probability of just about anything. The last two chapters have introduced models of syntactic constituency structure and its parsing. Here, we show that it is possible to build probabilistic models of syntactic knowledge and efficient probabilistic parsers.

One crucial use of probabilistic parsing is to solve the problem of **disambiguation**. Recall from Chapter 13 that sentences on average tend to be syntactically ambiguous because of phenomena like **coordination ambiguity** and **attachment ambiguity**. The CKY parsing algorithm can represent these ambiguities in an efficient way but is not equipped to resolve them. A probabilistic parser offers a solution to the problem: compute the probability of each interpretation and choose the most probable interpretation. The most commonly used probabilistic constituency grammar formalism is the **probabilistic context-free grammar** (PCFG), a probabilistic augmentation of context-free grammars in which each rule is associated with a probability. We introduce PCFGs in the next section, showing how they can be trained on Treebank grammars and how they can be parsed with a probabilistic version of the **CKY algorithm** of Chapter 13.

We then show a number of ways that we can improve on this basic probability model (PCFGs trained on Treebank grammars), such as by modifying the set of non-terminals (making them either more specific or more general), or adding more sophisticated conditioning factors like subcategorization or dependencies. Heavily lexicalized grammar formalisms such as Lexical-Functional Grammar (LFG) (Bresnan, 1982), Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994), Tree-Adjoining Grammar (TAG) (Joshi, 1985), and Combinatory Categorial Grammar (CCG) pose additional problems for probabilistic parsers. Section 14.7 introduces the task of **supertagging** and the use of heuristic search methods based on the **A* algorithm** in the context of CCG parsing.

Finally, we describe the standard techniques and metrics for evaluating parsers.

14.1 Probabilistic Context-Free Grammars

PCFG The simplest augmentation of the context-free grammar is the **Probabilistic Context-Free Grammar (PCFG)**, also known as the **Stochastic Context-Free Grammar (SCFG)**, first proposed by Booth (1969). Recall that a context-free grammar G is defined by four parameters (N, Σ, R, S) ; a probabilistic context-free grammar is also defined by four parameters, with a slight augmentation to each of the rules in R :

N	a set of non-terminal symbols (or variables)
Σ	a set of terminal symbols (disjoint from N)
R	a set of rules or productions, each of the form $A \rightarrow \beta [p]$, where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)^*$, and p is a number between 0 and 1 expressing $P(\beta A)$
S	a designated start symbol

That is, a PCFG differs from a standard CFG by augmenting each rule in R with a conditional probability:

$$A \rightarrow \beta [p] \quad (14.1)$$

Here p expresses the probability that the given non-terminal A will be expanded to the sequence β . That is, p is the conditional probability of a given expansion β given the left-hand-side (LHS) non-terminal A . We can represent this probability as

$$P(A \rightarrow \beta)$$

or as

$$P(A \rightarrow \beta | A)$$

or as

$$P(RHS | LHS)$$

Thus, if we consider all the possible expansions of a non-terminal, the sum of their probabilities must be 1:

$$\sum_{\beta} P(A \rightarrow \beta) = 1$$

Figure 14.1 shows a PCFG: a probabilistic augmentation of the \mathcal{L}_1 miniature English CFG grammar and lexicon. Note that the probabilities of all of the expansions of each non-terminal sum to 1. Also note that these probabilities were made up for pedagogical purposes. A real grammar has a great many more rules for each non-terminal; hence, the probabilities of any particular rule would tend to be much smaller.

consistent

A PCFG is said to be **consistent** if the sum of the probabilities of all sentences in the language equals 1. Certain kinds of recursive rules cause a grammar to be inconsistent by causing infinitely looping derivations for some sentences. For example, a rule $S \rightarrow S$ with probability 1 would lead to lost probability mass due to derivations that never terminate. See [Booth and Thompson \(1973\)](#) for more details on consistent and inconsistent grammars.

How are PCFGs used? A PCFG can be used to estimate a number of useful probabilities concerning a sentence and its parse tree(s), including the probability of a particular parse tree (useful in disambiguation) and the probability of a sentence or a piece of a sentence (useful in language modeling). Let's see how this works.

14.1.1 PCFGs for Disambiguation

A PCFG assigns a probability to each parse tree T (i.e., each **derivation**) of a sentence S . This attribute is useful in **disambiguation**. For example, consider the two parses of the sentence “Book the dinner flight” shown in Fig. 14.2. The sensible

Grammar		Lexicon
$S \rightarrow NP VP$	[.80]	$Det \rightarrow that [.10] a [.30] the [.60]$
$S \rightarrow Aux NP VP$	[.15]	$Noun \rightarrow book [.10] flight [.30]$
$S \rightarrow VP$	[.05]	$ meal [.05] money [.05]$
$NP \rightarrow Pronoun$	[.35]	$ flight [.40] dinner [.10]$
$NP \rightarrow Proper-Noun$	[.30]	$Verb \rightarrow book [.30] include [.30]$
$NP \rightarrow Det Nominal$	[.20]	$ prefer [.40]$
$NP \rightarrow Nominal$	[.15]	$Pronoun \rightarrow I [.40] she [.05]$
$Nominal \rightarrow Noun$	[.75]	$ me [.15] you [.40]$
$Nominal \rightarrow Nominal Noun$	[.20]	$Proper-Noun \rightarrow Houston [.60]$
$Nominal \rightarrow Nominal PP$	[.05]	$ NWA [.40]$
$VP \rightarrow Verb$	[.35]	$Aux \rightarrow does [.60] can [.40]$
$VP \rightarrow Verb NP$	[.20]	$Preposition \rightarrow from [.30] to [.30]$
$VP \rightarrow Verb NP PP$	[.10]	$ on [.20] near [.15]$
$VP \rightarrow Verb PP$	[.15]	$ through [.05]$
$VP \rightarrow Verb NP NP$	[.05]	
$VP \rightarrow VP PP$	[.15]	
$PP \rightarrow Preposition NP$	[1.0]	

Figure 14.1 A PCFG that is a probabilistic augmentation of the \mathcal{L}_1 miniature English CFG grammar and lexicon of Fig. 13.1. These probabilities were made up for pedagogical purposes and are not based on a corpus (since any real corpus would have many more rules, so the true probabilities of each rule would be much smaller).

parse on the left means “Book a flight that serves dinner”. The nonsensical parse on the right, however, would have to mean something like “Book a flight on behalf of ‘the dinner’” just as a structurally similar sentence like “Can you book John a flight?” means something like “Can you book a flight on behalf of John?”

The probability of a particular parse T is defined as the product of the probabilities of all the n rules used to expand each of the n non-terminal nodes in the parse tree T , where each rule i can be expressed as $LHS_i \rightarrow RHS_i$:

$$P(T, S) = \prod_{i=1}^n P(RHS_i | LHS_i) \quad (14.2)$$

The resulting probability $P(T, S)$ is both the joint probability of the parse and the sentence and also the probability of the parse $P(T)$. How can this be true? First, by the definition of joint probability:

$$P(T, S) = P(T)P(S|T) \quad (14.3)$$

But since a parse tree includes all the words of the sentence, $P(S|T)$ is 1. Thus,

$$P(T, S) = P(T)P(S|T) = P(T) \quad (14.4)$$

We can compute the probability of each of the trees in Fig. 14.2 by multiplying the probabilities of each of the rules used in the derivation. For example, the probability of the left tree in Fig. 14.2a (call it T_{left}) and the right tree (Fig. 14.2b or T_{right}) can be computed as follows:

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$

$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

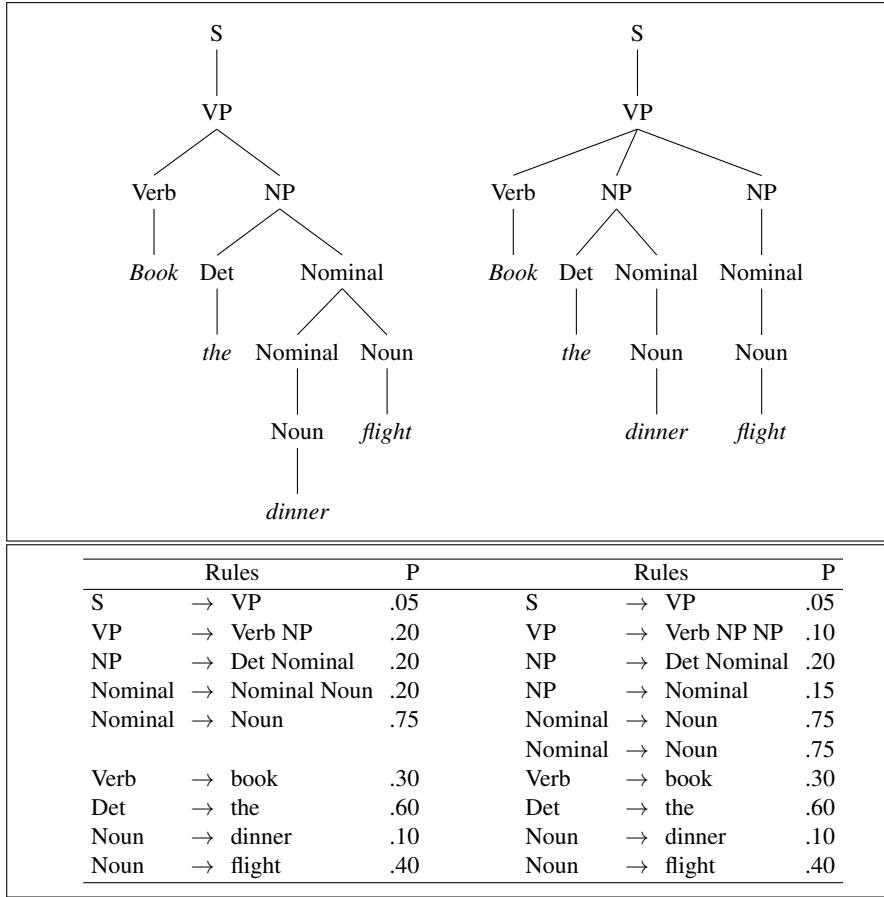


Figure 14.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the sensible meaning “Book a flight that serves dinner”, while the parse on the right corresponds to the nonsensical meaning “Book a flight on behalf of ‘the dinner’ ”.

We can see that the left tree in Fig. 14.2 has a much higher probability than the tree on the right. Thus, this parse would correctly be chosen by a disambiguation algorithm that selects the parse with the highest PCFG probability.

Let’s formalize this intuition that picking the parse with the highest probability is the correct way to do disambiguation. Consider all the possible parse trees for a given sentence S . The string of words S is called the **yield** of any parse tree over S . Thus, out of all parse trees with a yield of S , the disambiguation algorithm picks the parse tree that is most probable given S :

$$\hat{T}(S) = \underset{T: s.t. S = \text{yield}(T)}{\text{argmax}} P(T|S) \quad (14.5)$$

By definition, the probability $P(T|S)$ can be rewritten as $P(T, S)/P(S)$, thus leading to

$$\hat{T}(S) = \underset{T: s.t. S = \text{yield}(T)}{\text{argmax}} \frac{P(T, S)}{P(S)} \quad (14.6)$$

Since we are maximizing over all parse trees for the same sentence, $P(S)$ will be a

constant for each tree, so we can eliminate it:

$$\hat{T}(S) = \underset{T \text{ s.t. } S = \text{yield}(T)}{\operatorname{argmax}} P(T, S) \quad (14.7)$$

Furthermore, since we showed above that $P(T, S) = P(T)$, the final equation for choosing the most likely parse neatly simplifies to choosing the parse with the highest probability:

$$\hat{T}(S) = \underset{T \text{ s.t. } S = \text{yield}(T)}{\operatorname{argmax}} P(T) \quad (14.8)$$

14.1.2 PCFGs for Language Modeling

A second attribute of a PCFG is that it assigns a probability to the string of words constituting a sentence. This is important in **language modeling**, whether for use in speech recognition, machine translation, spelling correction, augmentative communication, or other applications. The probability of an unambiguous sentence is $P(T, S) = P(T)$ or just the probability of the single parse tree for that sentence. The probability of an ambiguous sentence is the sum of the probabilities of all the parse trees for the sentence:

$$P(S) = \sum_{T \text{ s.t. } S = \text{yield}(T)} P(T, S) \quad (14.9)$$

$$= \sum_{T \text{ s.t. } S = \text{yield}(T)} P(T) \quad (14.10)$$

An additional feature of PCFGs that is useful for language modeling is their ability to assign a probability to substrings of a sentence. For example, suppose we want to know the probability of the next word w_i in a sentence given all the words we've seen so far w_1, \dots, w_{i-1} . The general formula for this is

$$P(w_i | w_1, w_2, \dots, w_{i-1}) = \frac{P(w_1, w_2, \dots, w_{i-1}, w_i)}{P(w_1, w_2, \dots, w_{i-1})} \quad (14.11)$$

We saw in Chapter 3 a simple approximation of this probability using N -grams, conditioning on only the last word or two instead of the entire context; thus, the **bigram approximation** would give us

$$P(w_i | w_1, w_2, \dots, w_{i-1}) \approx \frac{P(w_{i-1}, w_i)}{P(w_{i-1})} \quad (14.12)$$

But the fact that the N -gram model can only make use of a couple words of context means it is ignoring potentially useful prediction cues. Consider predicting the word *after* in the following sentence from Chelba and Jelinek (2000):

(14.13) the contract ended with a loss of 7 cents after trading as low as 9 cents

A trigram grammar must predict *after* from the words *7 cents*, while it seems clear that the verb *ended* and the subject *contract* would be useful predictors that a PCFG-based parser could help us make use of. Indeed, it turns out that PCFGs allow us to condition on the entire previous context w_1, w_2, \dots, w_{i-1} shown in Eq. 14.11.

In summary, this section and the previous one have shown that PCFGs can be applied both to disambiguation in syntactic parsing and to word prediction in language modeling. Both of these applications require that we be able to compute the probability of parse tree T for a given sentence S . The next few sections introduce some algorithms for computing this probability.

14.2 Probabilistic CKY Parsing of PCFGs

The parsing problem for PCFGs is to produce the most-likely parse \hat{T} for a given sentence S , that is,

$$\hat{T}(S) = \underset{T: s.t. S = \text{yield}(T)}{\operatorname{argmax}} P(T) \quad (14.14)$$

probabilistic CKY

The algorithms for computing the most likely parse are simple extensions of the standard algorithms for parsing; most modern probabilistic parsers are based on the **probabilistic CKY** algorithm, first described by [Ney \(1991\)](#). The probabilistic CKY algorithm assumes the PCFG is in Chomsky normal form. Recall from page 222 that in CNF, the right-hand side of each rule must expand to either two non-terminals or to a single terminal, i.e., rules have the form $A \rightarrow B C$, or $A \rightarrow w$.

For the CKY algorithm, we represented each sentence as having indices between the words. Thus, an example sentence like

(14.15) Book the flight through Houston.

would assume the following indices between each word:

(14.16) ① Book ② the ③ flight ④ through ⑤ Houston ⑥

Using these indices, each constituent in the CKY parse tree is encoded in a two-dimensional matrix. Specifically, for a sentence of length n and a grammar that contains V non-terminals, we use the upper-triangular portion of an $(n+1) \times (n+1)$ matrix. For CKY, each cell $\text{table}[i, j]$ contained a list of constituents that could span the sequence of words from i to j . For probabilistic CKY, it's slightly simpler to think of the constituents in each cell as constituting a third dimension of maximum length V . This third dimension corresponds to each non-terminal that can be placed in this cell, and the value of the cell is then a probability for that non-terminal/constituent rather than a list of constituents. In summary, each cell $[i, j, A]$ in this $(n+1) \times (n+1) \times V$ matrix is the probability of a constituent of type A that spans positions i through j of the input.

Figure 14.3 gives the probabilistic CKY algorithm.

```

function PROBABILISTIC-CKY(words,grammar) returns most probable parse
    and its probability
    for  $j \leftarrow 1$  to LENGTH(words) do
        for all  $\{A \mid A \rightarrow \text{words}[j] \in \text{grammar}\}$ 
             $\text{table}[j-1, j, A] \leftarrow P(A \rightarrow \text{words}[j])$ 
        for  $i \leftarrow j-2$  downto 0 do
            for  $k \leftarrow i+1$  to  $j-1$  do
                for all  $\{A \mid A \rightarrow BC \in \text{grammar},$ 
                    and  $\text{table}[i, k, B] > 0$  and  $\text{table}[k, j, C] > 0\}$ 
                    if ( $\text{table}[i, j, A] < P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]$ ) then
                         $\text{table}[i, j, A] \leftarrow P(A \rightarrow BC) \times \text{table}[i, k, B] \times \text{table}[k, j, C]$ 
                         $\text{back}[i, j, A] \leftarrow \{k, B, C\}$ 
            return BUILD-TREE( $\text{back}[1, \text{LENGTH}(\text{words}), S]$ ),  $\text{table}[1, \text{LENGTH}(\text{words}), S]$ 

```

Figure 14.3 The probabilistic CKY algorithm for finding the maximum probability parse of a string of num_words words given a PCFG grammar with num_rules rules in Chomsky normal form. back is an array of backpointers used to recover the best parse. The *build_tree* function is left as an exercise to the reader.

Like the basic CKY algorithm in Fig. 13.5, the probabilistic CKY algorithm requires a grammar in Chomsky normal form. Converting a probabilistic grammar to CNF requires that we also modify the probabilities so that the probability of each parse remains the same under the new CNF grammar. Exercise 14.2 asks you to modify the algorithm for conversion to CNF in Chapter 13 so that it correctly handles rule probabilities.

In practice, a generalized CKY algorithm that handles unit productions directly is typically used. Recall that Exercise 13.3 asked you to make this change in CKY; Exercise 14.3 asks you to extend this change to probabilistic CKY.

Let's see an example of the probabilistic CKY chart, using the following mini-grammar, which is already in CNF:

$S \rightarrow NP VP$.80	$Det \rightarrow the$.40
$NP \rightarrow Det N$.30	$Det \rightarrow a$.40
$VP \rightarrow V NP$.20	$N \rightarrow meal$.01
$V \rightarrow includes$.05	$N \rightarrow flight$.02

Given this grammar, Fig. 14.4 shows the first steps in the probabilistic CKY parse of the sentence “The flight includes a meal”.

<i>The</i>	<i>flight</i>	<i>includes</i>	<i>a</i>	<i>meal</i>
Det: .40 [0,1]	NP: .30 * .40 * .02 = .0024 [0,2]	[0,3]	[0,4]	[0,5]
	N: .02 [1,2]	[1,3]	[1,4]	[1,5]
	V: .05 [2,3]	[2,4]	[2,5]	
		Det: .40 [3,4]	[3,5]	N: .01 [4,5]

Figure 14.4 The beginning of the probabilistic CKY matrix. Filling out the rest of the chart is left as Exercise 14.4 for the reader.

14.3 Ways to Learn PCFG Rule Probabilities

Where do PCFG rule probabilities come from? There are two ways to learn probabilities for the rules of a grammar. The simplest way is to use a treebank, a corpus of already parsed sentences. Recall that we introduced in Chapter 12 the idea of treebanks and the commonly used **Penn Treebank** (Marcus et al., 1993), a collection of parse trees in English, Chinese, and other languages that is distributed by the Linguistic Data Consortium. Given a treebank, we can compute the probability of each expansion of a non-terminal by counting the number of times that expansion occurs and then normalizing.

$$P(\alpha \rightarrow \beta | \alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_{\gamma} \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)} \quad (14.17)$$

If we don't have a treebank but we do have a (non-probabilistic) parser, we can generate the counts we need for computing PCFG rule probabilities by first parsing a corpus of sentences with the parser. If sentences were unambiguous, it would be as simple as this: parse the corpus, increment a counter for every rule in the parse, and then normalize to get probabilities.

But wait! Since most sentences are ambiguous, that is, have multiple parses, we don't know which parse to count the rules in. Instead, we need to keep a separate count for each parse of a sentence and weight each of these partial counts by the probability of the parse it appears in. But to get these parse probabilities to weight the rules, we need to already have a probabilistic parser.

inside-outside

The intuition for solving this chicken-and-egg problem is to incrementally improve our estimates by beginning with a parser with equal rule probabilities, then parse the sentence, compute a probability for each parse, use these probabilities to weight the counts, re-estimate the rule probabilities, and so on, until our probabilities converge. The standard algorithm for computing this solution is called the **inside-outside** algorithm; it was proposed by Baker (1979) as a generalization of the forward-backward algorithm for HMMs. Like forward-backward, inside-outside is a special case of the Expectation Maximization (EM) algorithm, and hence has two steps: the **expectation step**, and the **maximization step**. See Lari and Young (1990) or Manning and Schütze (1999) for more on the algorithm.

14.4 Problems with PCFGs

While probabilistic context-free grammars are a natural extension to context-free grammars, they have two main problems as probability estimators:

Poor independence assumptions: CFG rules impose an independence assumption on probabilities that leads to poor modeling of structural dependencies across the parse tree.

Lack of lexical conditioning: CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities.

Because of these problems, probabilistic constituent parsing models use some augmented version of PCFGs, or modify the Treebank-based grammar in some way.

In the next few sections after discussing the problems in more detail we introduce some of these augmentations.

14.4.1 Independence Assumptions Miss Rule Dependencies

Let's look at these problems in more detail. Recall that in a CFG the expansion of a non-terminal is independent of the context, that is, of the other nearby non-terminals in the parse tree. Similarly, in a PCFG, the probability of a particular rule like $NP \rightarrow Det\ N$ is also independent of the rest of the tree. By definition, the probability of a group of independent events is the product of their probabilities. These two facts explain why in a PCFG we compute the probability of a tree by just multiplying the probabilities of each non-terminal expansion.

Unfortunately, this CFG independence assumption results in poor probability estimates. This is because in English the choice of how a node expands can after all depend on the location of the node in the parse tree. For example, in English it turns out that *NPs* that are syntactic **subjects** are far more likely to be pronouns, and *NPs* that are syntactic **objects** are far more likely to be non-pronominal (e.g., a proper noun or a determiner noun sequence), as shown by these statistics for *NPs* in the Switchboard corpus (Francis et al., 1999):¹

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Unfortunately, there is no way to represent this contextual difference in the probabilities in a PCFG. Consider two expansions of the non-terminal *NP* as a pronoun or as a determiner+noun. How shall we set the probabilities of these two rules? If we set their probabilities to their overall probability in the Switchboard corpus, the two rules have about equal probability.

$$\begin{aligned} NP &\rightarrow DT\ NN & .28 \\ NP &\rightarrow PRP & .25 \end{aligned}$$

Because PCFGs don't allow a rule probability to be conditioned on surrounding context, this equal probability is all we get; there is no way to capture the fact that in subject position, the probability for $NP \rightarrow PRP$ should go up to .91, while in object position, the probability for $NP \rightarrow DT\ NN$ should go up to .66.

These dependencies could be captured if the probability of expanding an *NP* as a pronoun (e.g., $NP \rightarrow PRP$) versus a lexical *NP* (e.g., $NP \rightarrow DT\ NN$) were *conditioned* on whether the *NP* was a subject or an object. Section 14.5 introduces the technique of **parent annotation** for adding this kind of conditioning.

14.4.2 Lack of Sensitivity to Lexical Dependencies

A second class of problems with PCFGs is their lack of sensitivity to the words in the parse tree. Words do play a role in PCFGs since the parse probability includes the probability of a word given a part-of-speech (e.g., from rules like $V \rightarrow sleep$, $NN \rightarrow book$, etc.).

¹ Distribution of subjects from 31,021 declarative sentences; distribution of objects from 7,489 sentences. This tendency is caused by the use of subject position to realize the **topic** or old information in a sentence (Givón, 1990). Pronouns are a way to talk about old information, while non-pronominal ("lexical") noun-phrases are often used to introduce new referents (Chapter 22).

But it turns out that lexical information is useful in other places in the grammar, such as in resolving prepositional phrase (*PP*) attachment ambiguities. Since prepositional phrases in English can modify a noun phrase or a verb phrase, when a parser finds a prepositional phrase, it must decide where to attach it into the tree. Consider the following example:

(14.18) Workers dumped sacks into a bin.

Figure 14.5 shows two possible parse trees for this sentence; the one on the left is the correct parse; Fig. 14.6 shows another perspective on the preposition attachment problem, demonstrating that resolving the ambiguity in Fig. 14.5 is equivalent to deciding whether to attach the prepositional phrase into the rest of the tree at the *NP* or *VP* nodes; we say that the correct parse requires **VP attachment**, and the incorrect parse implies **NP attachment**.

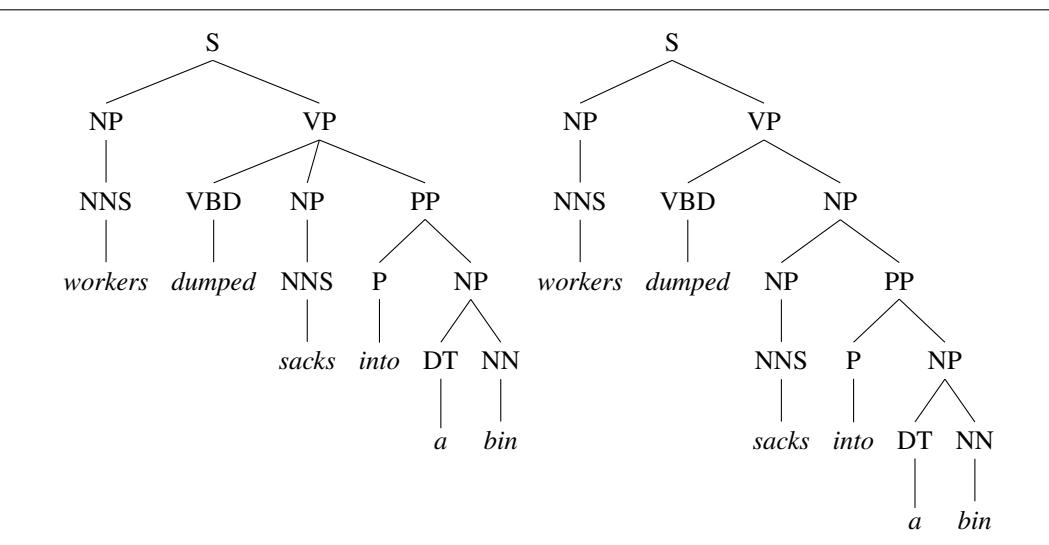


Figure 14.5 Two possible parse trees for a **prepositional phrase attachment ambiguity**. The left parse is the sensible one, in which “into a bin” describes the resulting location of the sacks. In the right incorrect parse, the sacks to be dumped are the ones which are already “into a bin”, whatever that might mean.

Why doesn’t a PCFG already deal with *PP* attachment ambiguities? Note that the two parse trees in Fig. 14.5 have almost exactly the same rules; they differ only in that the left-hand parse has this rule:

$$VP \rightarrow VBD\ NP\ PP$$

while the right-hand parse has these:

$$\begin{aligned} VP &\rightarrow VBD\ NP \\ NP &\rightarrow NP\ PP \end{aligned}$$

Depending on how these probabilities are set, a PCFG will **always** either prefer *NP* attachment or *VP* attachment. As it happens, *NP* attachment is slightly more common in English, so if we trained these rule probabilities on a corpus, we might always prefer *NP* attachment, causing us to misparse this sentence.

But suppose we set the probabilities to prefer the *VP* attachment for this sentence. Now we would misparse the following, which requires *NP* attachment:



Figure 14.6 Another view of the preposition attachment problem. Should the *PP* on the right attach to the *VP* or *NP* nodes of the partial parse tree on the left?

(14.19) fishermen caught tons of herring

What information in the input sentence lets us know that (14.19) requires *NP* attachment while (14.18) requires *VP* attachment? These preferences come from the identities of the verbs, nouns, and prepositions. The affinity between the verb *dumped* and the preposition *into* is greater than the affinity between the noun *sacks* and the preposition *into*, thus leading to *VP* attachment. On the other hand, in (14.19) the affinity between *tons* and *of* is greater than that between *caught* and *of*, leading to *NP* attachment. Thus, to get the correct parse for these kinds of examples, we need a model that somehow augments the PCFG probabilities to deal with these **lexical dependency** statistics for different verbs and prepositions.

lexical dependency

Coordination ambiguities are another case in which lexical dependencies are the key to choosing the proper parse. Figure 14.7 shows an example from Collins (1999) with two parses for the phrase *dogs in houses and cats*. Because *dogs* is semantically a better conjunct for *cats* than *houses* (and because most dogs can't fit inside cats), the parse *[dogs in [NP houses and cats]]* is intuitively unnatural and should be dispreferred. The two parses in Fig. 14.7, however, have exactly the same PCFG rules, and thus a PCFG will assign them the same probability.

In summary, we have shown in this section and the previous one that probabilistic context-free grammars are incapable of modeling important **structural** and **lexical** dependencies. In the next two sections we sketch current methods for augmenting PCFGs to deal with both these issues.

14.5 Improving PCFGs by Splitting Non-Terminals

Let's start with the first of the two problems with PCFGs mentioned above: their inability to model structural dependencies, like the fact that NPs in subject position tend to be pronouns, whereas NPs in object position tend to have full lexical (non-pronominal) form. How could we augment a PCFG to correctly model this fact?

split

One idea would be to **split** the *NP* non-terminal into two versions: one for subjects, one for objects. Having two nodes (e.g., *NP_{subject}* and *NP_{object}*) would allow us to correctly model their different distributional properties, since we would have



Figure 14.7 An instance of coordination ambiguity. Although the left structure is intuitively the correct one, a PCFG will assign them identical probabilities since both structures use exactly the same set of rules. After [Collins \(1999\)](#).

different probabilities for the rule $NP_{subject} \rightarrow PRP$ and the rule $NP_{object} \rightarrow PRP$.

parent annotation
 One way to implement this intuition of splits is to do **parent annotation** (Johnson, 1998), in which we annotate each node with its parent in the parse tree. Thus, an NP node that is the subject of the sentence and hence has parent S would be annotated NP^S , while a direct object NP whose parent is VP would be annotated NP^VP . Figure 14.8 shows an example of a tree produced by a grammar that parent-annotates the phrasal non-terminals (like NP and VP).

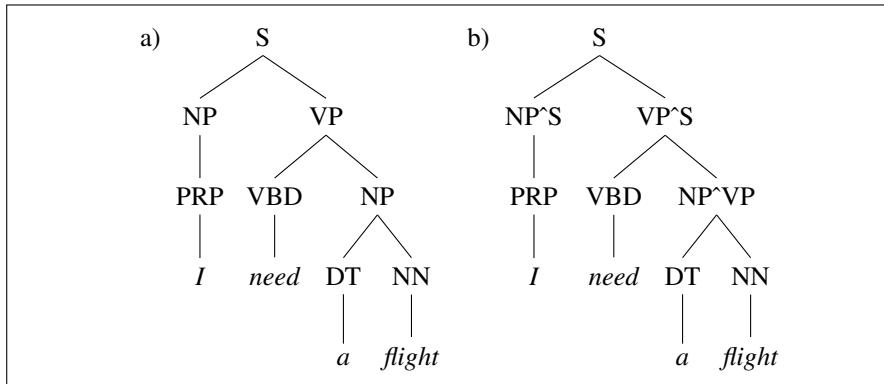


Figure 14.8 A standard PCFG parse tree (a) and one which has **parent annotation** on the nodes which aren't pre-terminal (b). All the non-terminal nodes (except the pre-terminal part-of-speech nodes) in parse (b) have been annotated with the identity of their parent.

In addition to splitting these phrasal nodes, we can also improve a PCFG by splitting the pre-terminal part-of-speech nodes (Klein and Manning, 2003b). For example, different kinds of adverbs (RB) tend to occur in different syntactic positions: the most common adverbs with $ADVP$ parents are *also* and *now*, with VP parents *n't* and *not*, and with NP parents *only* and *just*. Thus, adding tags like RB^ADVP , RB^VP , and RB^NP can be useful in improving PCFG modeling.

Similarly, the Penn Treebank tag IN can mark a wide variety of parts-of-speech, including subordinating conjunctions (*while*, *as*, *if*), complementizers (*that*, *for*), and prepositions (*of*, *in*, *from*). Some of these differences can be captured by parent

annotation (subordinating conjunctions occur under S, prepositions under PP), while others require splitting the pre-terminal nodes. Figure 14.9 shows an example from Klein and Manning (2003b) in which even a parent-annotated grammar incorrectly parses *works* as a noun in *to see if advertising works*. Splitting pre-terminals to allow *if* to prefer a sentential complement results in the correct verbal parse.

split and merge

Node-splitting is not without problems; it increases the size of the grammar and hence reduces the amount of training data available for each grammar rule, leading to overfitting. Thus, it is important to split to just the correct level of granularity for a particular training set. While early models employed handwritten rules to try to find an optimal number of non-terminals (Klein and Manning, 2003b), modern models automatically search for the optimal splits. The **split and merge** algorithm of Petrov et al. (2006), for example, starts with a simple X-bar grammar, alternately splits the non-terminals, and merges non-terminals, finding the set of annotated nodes that maximizes the likelihood of the training set treebank.

14.6 Probabilistic Lexicalized CFGs

The previous section showed that a simple probabilistic CKY algorithm for parsing raw PCFGs can achieve extremely high parsing accuracy if the grammar rule symbols are redesigned by automatic splits and merges.

In this section, we discuss an alternative family of models in which instead of modifying the grammar rules, we modify the probabilistic model of the parser to allow for **lexicalized** rules. The resulting family of lexicalized parsers includes the **Collins parser** (Collins, 1999) and the **Charniak parser** (Charniak, 1997).

lexicalized grammar

We saw in Section 12.4.3 that syntactic constituents could be associated with a lexical **head**, and we defined a **lexicalized grammar** in which each non-terminal in the tree is annotated with its lexical head, where a rule like $VP \rightarrow VBD\ NP\ PP$

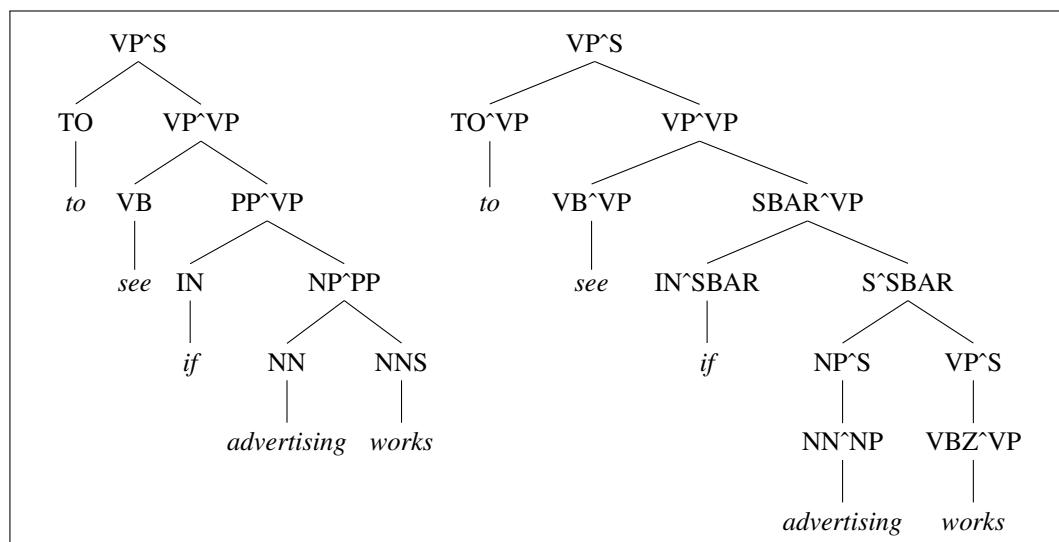


Figure 14.9 An incorrect parse even with a parent-annotated parse (left). The correct parse (right), was produced by a grammar in which the pre-terminal nodes have been split, allowing the probabilistic grammar to capture the fact that *if* prefers sentential complements. Adapted from Klein and Manning (2003b).

would be extended as

$$VP(dumped) \rightarrow VBD(dumped) \ NP(sacks) \ PP(into) \quad (14.20)$$

In the standard type of lexicalized grammar, we actually make a further extension, which is to associate the **head tag**, the part-of-speech tags of the headwords, with the non-terminal symbols as well. Each rule is thus lexicalized by both the headword and the head tag of each constituent resulting in a format for lexicalized rules like

$$VP(dumped,VBD) \rightarrow VBD(dumped,VBD) \ NP(sacks,NNS) \ PP(into,P) \quad (14.21)$$

We show a lexicalized parse tree with head tags in Fig. 14.10, extended from Fig. 12.11.



Figure 14.10 A lexicalized tree, including head tags, for a WSJ sentence, adapted from Collins (1999). Below we show the PCFG rules needed for this parse tree, internal rules on the left, and lexical rules on the right.

To generate such a lexicalized tree, each PCFG rule must be augmented to identify one right-hand constituent to be the head daughter. The headword for a node is then set to the headword of its head daughter, and the head tag to the part-of-speech tag of the headword. Recall that we gave in Fig. 12.12 a set of handwritten rules for identifying the heads of particular constituents.

A natural way to think of a lexicalized grammar is as a parent annotation, that is, as a simple context-free grammar with many copies of each rule, one copy for each possible headword/head tag for each constituent. Thinking of a probabilistic lexicalized CFG in this way would lead to the set of simple PCFG rules shown below the tree in Fig. 14.10.

lexical rules
internal rules

Note that Fig. 14.10 shows two kinds of rules: **lexical rules**, which express the expansion of a pre-terminal to a word, and **internal rules**, which express the

other rule expansions. We need to distinguish these kinds of rules in a lexicalized grammar because they are associated with very different kinds of probabilities. The lexical rules are deterministic, that is, they have probability 1.0 since a lexicalized pre-terminal like $NN(bin,NN)$ can only expand to the word *bin*. But for the internal rules, we need to estimate probabilities.

Suppose we were to treat a probabilistic lexicalized CFG like a really big CFG that just happened to have lots of very complex non-terminals and estimate the probabilities for each rule from maximum likelihood estimates. Thus, according to Eq. 14.17, the MLE estimate for the probability for the rule $P(VP(dumped,VBD) \rightarrow VBD(dumped, VBD) NP(sacks,NNS) PP(into,P))$ would be

$$\frac{Count(VP(dumped,VBD) \rightarrow VBD(dumped, VBD) NP(sacks,NNS) PP(into,P))}{Count(VP(dumped,VBD))} \quad (14.22)$$

But there's no way we can get good estimates of counts like those in (14.22) because they are so specific: we're unlikely to see many (or even any) instances of a sentence with a verb phrase headed by *dumped* that has one *NP* argument headed by *sacks* and a *PP* argument headed by *into*. In other words, counts of fully lexicalized PCFG rules like this will be far too sparse, and most rule probabilities will come out 0.

The idea of lexicalized parsing is to make some further independence assumptions to break down each rule so that we would estimate the probability

$$P(VP(dumped,VBD) \rightarrow VBD(dumped, VBD) NP(sacks,NNS) PP(into,P))$$

as the product of smaller independent probability estimates for which we could acquire reasonable counts. The next section summarizes one such method, the Collins parsing method.

14.6.1 The Collins Parser

Statistical parsers differ in exactly which independence assumptions they make. Let's look at the assumptions in a simplified version of the Collins parser. The first intuition of the Collins parser is to think of the right-hand side of every (internal) CFG rule as consisting of a head non-terminal, together with the non-terminals to the left of the head and the non-terminals to the right of the head. In the abstract, we think about these rules as follows:

$$LHS \rightarrow L_n L_{n-1} \dots L_1 H R_1 \dots R_{n-1} R_n \quad (14.23)$$

Since this is a lexicalized grammar, each of the symbols like L_1 or R_3 or H or LHS is actually a complex symbol representing the category and its head and head tag, like $VP(dumped, VP)$ or $NP(sacks, NNS)$.

Now, instead of computing a single MLE probability for this rule, we are going to break down this rule via a neat generative story, a slight simplification of what is called Collins Model 1. This new generative story is that given the left-hand side, we first generate the head of the rule and then generate the dependents of the head, one by one, from the inside out. Each of these steps will have its own probability.

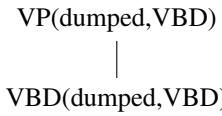
We also add a special STOP non-terminal at the left and right edges of the rule; this non-terminal allows the model to know when to stop generating dependents on a given side. We generate dependents on the left side of the head until we've generated STOP on the left side of the head, at which point we move to the right side of the head and start generating dependents there until we generate STOP. So it's as if we

are generating a rule augmented as follows:

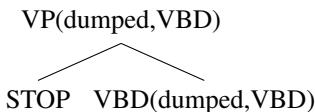
$$\begin{aligned} P(VP(dumped, VBD) \rightarrow & \\ \text{STOP } VBD(dumped, VBD) \text{ } NP(sacks, NNS) \text{ } PP(into, P) \text{ } STOP) & \end{aligned} \quad (14.24)$$

Let's see the generative story for this augmented rule. We make use of three kinds of probabilities: P_H for generating heads, P_L for generating dependents on the left, and P_R for generating dependents on the right.

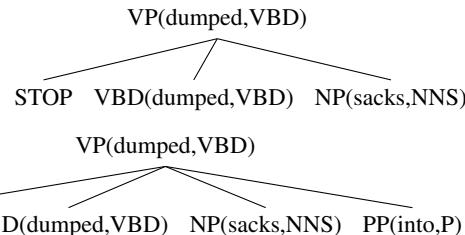
1. Generate the head $VBD(dumped, VBD)$ with probability
 $P(H|LHS) = P(VBD(dumped, VBD) | VP(dumped, VBD))$



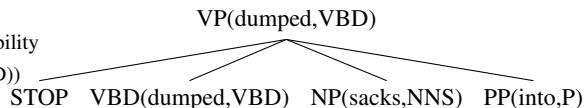
2. Generate the left dependent (which is $STOP$, since there isn't one) with probability
 $P(STOP | VP(dumped, VBD) \text{ } VBD(dumped, VBD))$



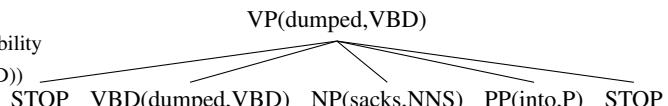
3. Generate right dependent $NP(sacks, NNS)$ with probability
 $P_r(NP(sacks, NNS) | VP(dumped, VBD), VBD(dumped, VBD))$



4. Generate the right dependent $PP(into, P)$ with probability
 $P_r(PP(into, P) | VP(dumped, VBD), VBD(dumped, VBD))$



- 5) Generate the right dependent $STOP$ with probability
 $P_r(STOP | VP(dumped, VBD), VBD(dumped, VBD))$



In summary, the probability of this rule

$$\begin{aligned} P(VP(dumped, VBD) \rightarrow & \\ \text{STOP } VBD(dumped, VBD) \text{ } NP(sacks, NNS) \text{ } PP(into, P)) & \end{aligned} \quad (14.25)$$

is estimated as

$$\begin{aligned} P_H(VBD | VP, dumped) &\times P_L(STOP | VP, VBD, dumped) \\ &\times P_R(NP(sacks, NNS) | VP, VBD, dumped) \\ &\times P_R(PP(into, P) | VP, VBD, dumped) \\ &\times P_R(STOP | VP, VBD, dumped) \end{aligned} \quad (14.26)$$

Each of these probabilities can be estimated from much smaller amounts of data than the full probability in (14.25). For example, the maximum likelihood estimate for the component probability $P_R(NP(sacks, NNS) | VP, VBD, dumped)$ is

$$\frac{\text{Count}(VP(dumped, VBD) \text{ with } NNS(sacks) \text{ as a daughter somewhere on the right})}{\text{Count}(VP(dumped, VBD))} \quad (14.27)$$

These counts are much less subject to sparsity problems than are complex counts like those in (14.25).

More generally, if H is a head with head word hw and head tag ht , lw/lt and rw/rt are the word/tag on the left and right respectively, and P is the parent, then the probability of an entire rule can be expressed as follows:

1. Generate the head of the phrase $H(hw, ht)$ with probability:

$$P_H(H(hw, ht) | P, hw, ht)$$

2. Generate modifiers to the left of the head with total probability

$$\prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i) | P, H, hw, ht)$$

such that $L_{n+1}(lw_{n+1}, lt_{n+1}) = \text{STOP}$, and we stop generating once we've generated a STOP token.

3. Generate modifiers to the right of the head with total probability:

$$\prod_{i=1}^{n+1} P_R(R_i(rt_i, rw_i) | P, H, hw, ht)$$

such that $R_{n+1}(rt_{n+1}, rw_{n+1}) = \text{STOP}$, and we stop generating once we've generated a STOP token.

The parsing algorithm for the Collins model is an extension of probabilistic CKY. Extending the CKY algorithm to handle basic lexicalized probabilities is left as Exercises 14.5 and 14.6 for the reader.

14.7 Probabilistic CCG Parsing

Lexicalized grammar frameworks such as CCG pose problems for which the phrase-based methods we've been discussing are not particularly well-suited. To quickly review, CCG consists of three major parts: a set of categories, a lexicon that associates words with categories, and a set of rules that govern how categories combine in context. Categories can be either atomic elements, such as S and NP , or functions such as $(S \setminus NP)/NP$ which specifies the transitive verb category. Rules specify how functions, their arguments, and other functions combine. For example, the following rule templates, **forward** and **backward function application**, specify the way that functions apply to their arguments.

$$\begin{aligned} X/Y \ Y &\Rightarrow X \\ Y \ X \setminus Y &\Rightarrow X \end{aligned}$$

The first rule applies a function to its argument on the right, while the second looks to the left for its argument. The result of applying either of these rules is the category specified as the value of the function being applied. For the purposes of this discussion, we'll rely on these two rules along with the **forward** and **backward composition** rules and **type-raising**, as described in Chapter 12.

14.7.1 Ambiguity in CCG

As is always the case in parsing, managing ambiguity is the key to successful CCG parsing. The difficulties with CCG parsing arise from the ambiguity caused by the large number of complex lexical categories combined with the very general nature of the grammatical rules. To see some of the ways that ambiguity arises in a categorial framework, consider the following example.

(14.28) United diverted the flight to Reno.

Our grasp of the role of *the flight* in this example depends on whether the prepositional phrase *to Reno* is taken as a modifier of *the flight*, as a modifier of the entire verb phrase, or as a potential second argument to the verb *divert*. In a context-free grammar approach, this ambiguity would manifest itself as a choice among the following rules in the grammar.

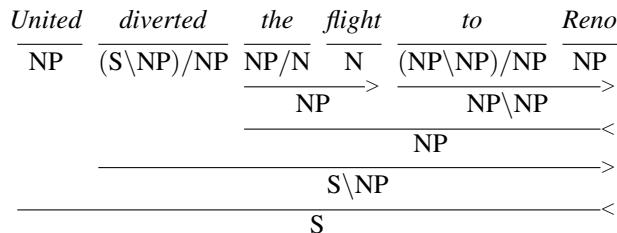
$$\text{Nominal} \rightarrow \text{Nominal PP}$$

$$\text{VP} \rightarrow \text{VP PP}$$

$$\text{VP} \rightarrow \text{Verb NP PP}$$

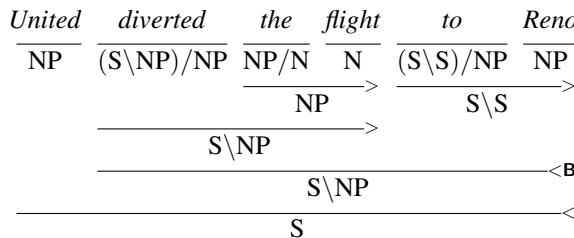
In a phrase-structure approach we would simply assign the word *to* to the category *P* allowing it to combine with *Reno* to form a prepositional phrase. The subsequent choice of grammar rules would then dictate the ultimate derivation. In the categorial approach, we can associate *to* with distinct categories to reflect the ways in which it might interact with other elements in a sentence. The fairly abstract combinatoric rules would then sort out which derivations are possible. Therefore, the source of ambiguity arises not from the grammar but rather from the lexicon.

Let's see how this works by considering several possible derivations for this example. To capture the case where the prepositional phrase *to Reno* modifies *the flight*, we assign the preposition *to* the category $(NP \setminus NP)/NP$, which gives rise to the following derivation.

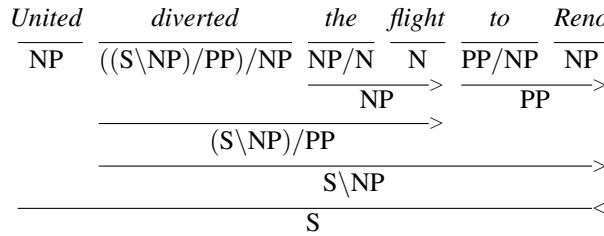


Here, the category assigned to *to* expects to find two arguments: one to the right as with a traditional preposition, and one to the left that corresponds to the *NP* to be modified.

Alternatively, we could assign *to* to the category $(S \setminus S)/NP$, which permits the following derivation where *to Reno* modifies the preceding verb phrase.



A third possibility is to view *divert* as a ditransitive verb by assigning it to the category $((S \setminus NP)/PP)/NP$, while treating *to Reno* as a simple prepositional phrase.



While CCG parsers are still subject to ambiguity arising from the choice of grammar rules, including the kind of spurious ambiguity discussed in Chapter 12, it should be clear that the choice of lexical categories is the primary problem to be addressed in CCG parsing.

14.7.2 CCG Parsing Frameworks

Since the rules in combinatory grammars are either binary or unary, a bottom-up, tabular approach based on the CKY algorithm should be directly applicable to CCG parsing. Recall from Fig. 14.3 that PCKY employs a table that records the location, category and probability of all valid constituents discovered in the input. Given an appropriate probability model for CCG derivations, the same kind of approach can work for CCG parsing.

Unfortunately, the large number of lexical categories available for each word, combined with the promiscuity of CCG’s combinatoric rules, leads to an explosion in the number of (mostly useless) constituents added to the parsing table. The key to managing this explosion of zombie constituents is to accurately assess and exploit the most likely lexical categories possible for each word — a process called supertagging.

The following sections describe two approaches to CCG parsing that make use of supertags. Section 14.7.4, presents an approach that structures the parsing process as a heuristic search through the use of the A* algorithm. The following section then briefly describes a more traditional maximum entropy approach that manages the search space complexity through the use of **adaptive supertagging** — a process that iteratively considers more and more tags until a parse is found.

14.7.3 Supertagging

supertagging Chapter 8 introduced the task of part-of-speech tagging, the process of assigning the correct lexical category to each word in a sentence. **Supertagging** is the corresponding task for highly lexicalized grammar frameworks, where the assigned tags often dictate much of the derivation for a sentence.

CCG supertaggers rely on treebanks such as CCGbank to provide both the overall set of lexical categories as well as the allowable category assignments for each word in the lexicon. CCGbank includes over 1000 lexical categories, however, in practice, most supertaggers limit their tagsets to those tags that occur at least 10 times in the training corpus. This results in an overall total of around 425 lexical categories available for use in the lexicon. Note that even this smaller number is large in contrast to the 45 POS types used by the Penn Treebank tagset.

As with traditional part-of-speech tagging, the standard approach to building a CCG supertagger is to use supervised machine learning to build a sequence classifier using labeled training data. A common approach is to use the maximum entropy Markov model (MEMM), as described in Chapter 8, to find the most likely sequence of tags given a sentence. The features in such a model consist of the current word w_i , its surrounding words within l words w_{i-l}^{i+l} , as well as the k previously assigned supertags t_{i-k}^{i-1} . This type of model is summarized in the following equation from Chapter 8. Training by maximizing log-likelihood of the training corpus and decoding via the Viterbi algorithm are the same as described in Chapter 8.

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \prod_i P(t_i|w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\ &= \operatorname{argmax}_T \prod_i \frac{\exp\left(\sum_i w_i f_i(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_i w_i f_i(t', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}\end{aligned}\quad (14.29)$$

Word and tag-based features with k and l both set to 2 provides reasonable results given sufficient training data. Additional features such as POS tags and short character suffixes are also commonly used to improve performance.

Unfortunately, even with additional features the large number of possible supertags combined with high per-word ambiguity leads to error rates that are too high for practical use in a parser. More specifically, the single best tag sequence \hat{T} will typically contain too many incorrect tags for effective parsing to take place. To overcome this, we can instead return a probability distribution over the possible supertags for each word in the input. The following table illustrates an example distribution for a simple example sentence. In this table, each column represents the probability of each supertag for a given word *in the context of the input sentence*. The “...” represent all the remaining supertags possible for each word.

United	serves	Denver
$N/N: 0.4$	$(S \setminus NP)/NP: 0.8$	$NP: 0.9$
$NP: 0.3$	$N: 0.1$	$N/N: 0.05$
$S/S: 0.1$
$S \setminus S: .05$		
...		

In a MEMM framework, the probability of the optimal tag sequence defined in Eq. 14.29 is efficiently computed with a suitably modified version of the Viterbi algorithm. However, since Viterbi only finds the single best tag sequence it doesn't provide exactly what we need here; we need to know the probability of each possible word/tag pair. The probability of any given tag for a word is the sum of the probabilities of all the supertag sequences that contain that tag at that location. A table representing these values can be computed efficiently by using a version of the forward-backward algorithm used for HMMs.

The same result can also be achieved through recurrent neural network (RNN) sequence models, which have the advantage of embeddings to represent inputs and allow representations that span the entire sentence, as opposed to size-limited sliding

windows. RNN approaches also avoid the use of high-level features, such as part of speech tags, which is helpful since errors in tag assignment can propagate to errors in supertags. As with the forward-backward algorithm, RNN-based methods can provide a probability distribution over the lexical categories for each word in the input.

14.7.4 CCG Parsing using the A* Algorithm

The A* algorithm is a heuristic search method that employs an agenda to find an optimal solution. Search states representing partial solutions are added to an agenda based on a cost function, with the least-cost option being selected for further exploration at each iteration. When a state representing a complete solution is first selected from the agenda, it is guaranteed to be optimal and the search terminates.

The A* cost function, $f(n)$, is used to efficiently guide the search to a solution. The f -cost has two components: $g(n)$, the exact cost of the partial solution represented by the state n , and $h(n)$ a heuristic approximation of the cost of a solution that makes use of n . When $h(n)$ satisfies the criteria of not overestimating the actual cost, A* will find an optimal solution. Not surprisingly, the closer the heuristic can get to the actual cost, the more effective A* is at finding a solution without having to explore a significant portion of the solution space.

When applied to parsing, search states correspond to edges representing completed constituents. As with the PCKY algorithm, edges specify a constituent's start and end positions, its grammatical category, and its f -cost. Here, the g component represents the current cost of an edge and the h component represents an estimate of the cost to complete a derivation that makes use of that edge. The use of A* for phrase structure parsing originated with (Klein and Manning, 2003a), while the CCG approach presented here is based on (Lewis and Steedman, 2014).

Using information from a supertagger, an agenda and a parse table are initialized with states representing all the possible lexical categories for each word in the input, along with their f -costs. The main loop removes the lowest cost edge from the agenda and tests to see if it is a complete derivation. If it reflects a complete derivation it is selected as the best solution and the loop terminates. Otherwise, new states based on the applicable CCG rules are generated, assigned costs, and entered into the agenda to await further processing. The loop continues until a complete derivation is discovered, or the agenda is exhausted, indicating a failed parse. The algorithm is given in Fig. 14.11.

Heuristic Functions

Before we can define a heuristic function for our A* search, we need to decide how to assess the quality of CCG derivations. For the generic PCFG model, we defined the probability of a tree as the product of the probability of the rules that made up the tree. Given CCG's lexical nature, we'll make the simplifying assumption that the probability of a CCG derivation is just the product of the probability of the supertags assigned to the words in the derivation, ignoring the rules used in the derivation. More formally, given a sentence S and derivation D that contains supertag sequence T , we have:

$$P(D, S) = P(T, S) \quad (14.30)$$

$$= \prod_{i=1}^n P(t_i | s_i) \quad (14.31)$$

```

function CCG-ASTAR-PARSE(words) returns table or failure
  supertags  $\leftarrow$  SUPERTAGGER(words)
  for i  $\leftarrow$  from 1 to LENGTH(words) do
    for all {A | (words[i], A, score)  $\in$  supertags} do
      edge  $\leftarrow$  MAKEEDGE(i - 1, i, A, score)
      table  $\leftarrow$  INSERTEDGE(table, edge)
      agenda  $\leftarrow$  INSERTEDGE(agenda, edge)
    loop do
      if EMPTY?(agenda) return failure
      current  $\leftarrow$  POP(agenda)
      if COMPLETEDPARSE?(current) return table
      table  $\leftarrow$  INSERTEDGE(chart, edge)
      for each rule in APPLICABLERULES(edge) do
        successor  $\leftarrow$  APPLY(rule, edge)
        if successor not  $\in$  agenda or chart
          agenda  $\leftarrow$  INSERTEDGE(agenda, successor)
        else if successor  $\in$  agenda with higher cost
          agenda  $\leftarrow$  REPLACEEDGE(agenda, successor)

```

Figure 14.11 A*-based CCG parsing.

To better fit with the traditional A* approach, we'd prefer to have states scored by a cost function where lower is better (i.e., we're trying to minimize the cost of a derivation). To achieve this, we'll use negative log probabilities to score derivations; this results in the following equation, which we'll use to score completed CCG derivations.

$$P(D, S) = P(T, S) \quad (14.32)$$

$$= \sum_{i=1}^n -\log P(t_i | s_i) \quad (14.33)$$

Given this model, we can define our *f*-cost as follows. The *f*-cost of an edge is the sum of two components: *g*(*n*), the cost of the span represented by the edge, and *h*(*n*), the estimate of the cost to complete a derivation containing that edge (these are often referred to as the **inside** and **outside costs**). We'll define *g*(*n*) for an edge using Equation 14.33. That is, it is just the sum of the costs of the supertags that comprise the span.

For *h*(*n*), we need a score that approximates but *never overestimates* the actual cost of the final derivation. A simple heuristic that meets this requirement assumes that each of the words in the outside span will be assigned its *most probable supertag*. If these are the tags used in the final derivation, then its score will equal the heuristic. If any other tags are used in the final derivation the *f*-cost will be higher since the new tags must have higher costs, thus guaranteeing that we will not overestimate.

Putting this all together, we arrive at the following definition of a suitable *f*-cost

for an edge.

$$\begin{aligned}
 f(w_{i,j}, t_{i,j}) &= g(w_{i,j}) + h(w_{i,j}) \\
 &= \sum_{k=i}^j -\log P(t_k | w_k) + \\
 &\quad \sum_{k=1}^{i-1} \min_{t \in \text{tags}} (-\log P(t | w_k)) + \sum_{k=j+1}^N \min_{t \in \text{tags}} (-\log P(t | w_k))
 \end{aligned} \tag{14.34}$$

As an example, consider an edge representing the word *serves* with the supertag *N* in the following example.

(14.35) United serves Denver.

The *g*-cost for this edge is just the negative log probability of this tag, $-\log_{10}(0.1)$, or 1. The outside *h*-cost consists of the most optimistic supertag assignments for *United* and *Denver*, which are *N/N* and *NP* respectively. The resulting *f*-cost for this edge is therefore 1.443.

An Example

Fig. 14.12 shows the initial agenda and the progress of a complete parse for this example. After initializing the agenda and the parse table with information from the supertagger, it selects the best edge from the agenda — the entry for *United* with the tag *N/N* and *f*-cost 0.591. This edge does not constitute a complete parse and is therefore used to generate new states by applying all the relevant grammar rules. In this case, applying forward application to *United: N/N* and *serves: N* results in the creation of the edge *United serves: N[0,2], 1.795* to the agenda.

Skipping ahead, at the third iteration an edge representing the complete derivation *United serves Denver, S[0,3], .716* is added to the agenda. However, the algorithm does not terminate at this point since the cost of this edge (.716) does not place it at the top of the agenda. Instead, the edge representing *Denver* with the category *NP* is popped. This leads to the addition of another edge to the agenda (type-raising *Denver*). Only after this edge is popped and dealt with does the earlier state representing a complete derivation rise to the top of the agenda where it is popped, goal tested, and returned as a solution.

The effectiveness of the A* approach is reflected in the coloring of the states in Fig. 14.12 as well as the final parsing table. The edges shown in blue (including all the initial lexical category assignments not explicitly shown) reflect states in the search space that never made it to the top of the agenda and, therefore, never contributed any edges to the final table. This is in contrast to the PCKY approach where the parser systematically fills the parse table with all possible constituents for all possible spans in the input, filling the table with myriad constituents that do not contribute to the final analysis.

14.8 Evaluating Parsers

The standard techniques for evaluating parsers and grammars are called the PARSEVAL measures; they were proposed by Black et al. (1991) and were based on the same ideas from signal-detection theory that we saw in earlier chapters. The

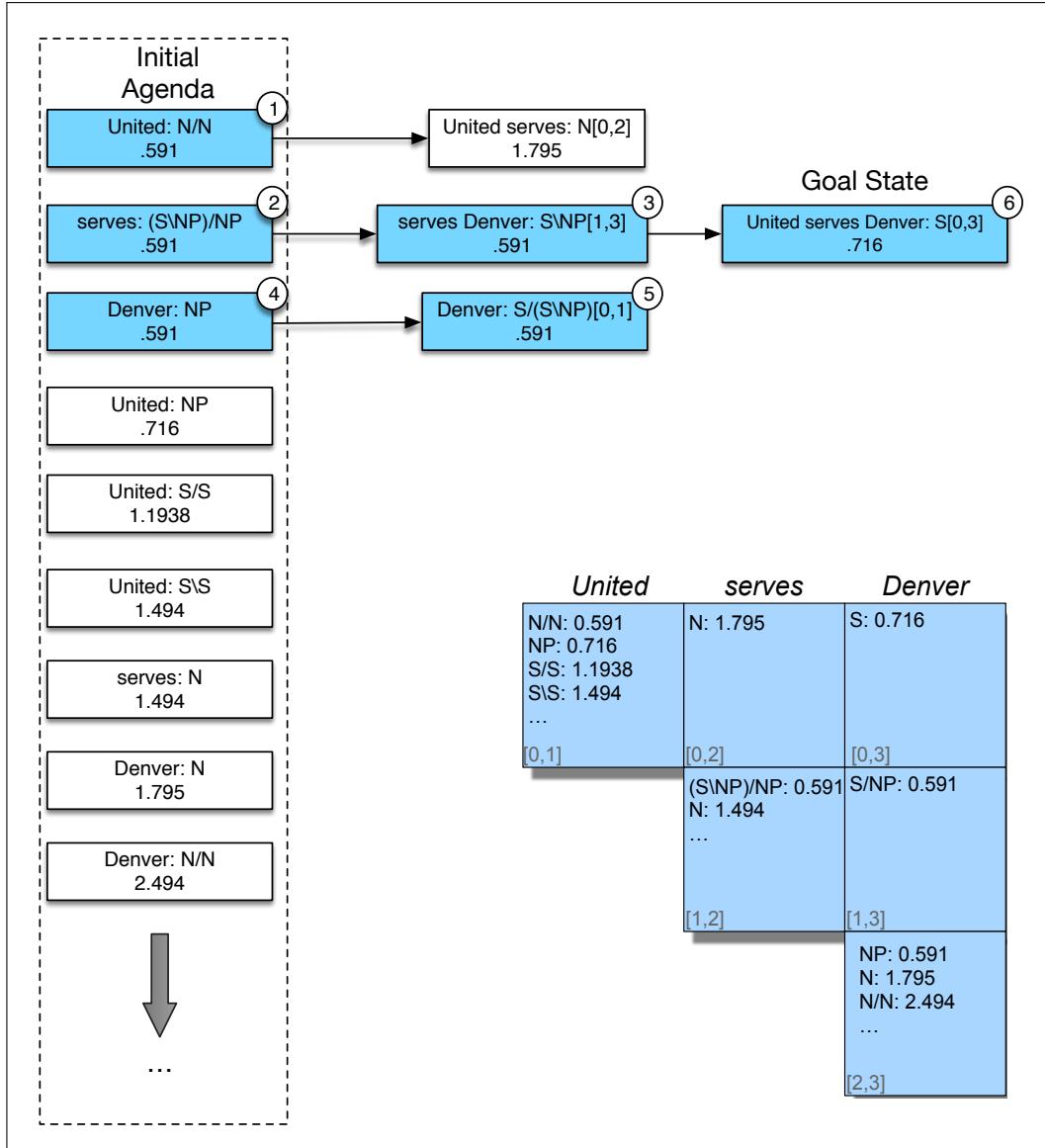


Figure 14.12 Example of an A* search for the example “United serves Denver”. The circled numbers on the blue boxes indicate the order in which the states are popped from the agenda. The costs in each state are based on f-costs using negative \log_{10} probabilities.

intuition of the PARSEVAL metric is to measure how much the **constituents** in the hypothesis parse tree look like the constituents in a hand-labeled, gold-reference parse. PARSEVAL thus assumes we have a human-labeled “gold standard” parse tree for each sentence in the test set; we generally draw these gold-standard parses from a treebank like the Penn Treebank.

Given these gold-standard reference parses for a test set, a given constituent in a hypothesis parse C_h of a sentence s is labeled “correct” if there is a constituent in the reference parse C_r with the same starting point, ending point, and non-terminal symbol. We can then measure the precision and recall just as we did for chunking in the previous chapter.

$$\text{labeled recall: } = \frac{\# \text{ of correct constituents in hypothesis parse of } s}{\# \text{ of correct constituents in reference parse of } s}$$

$$\text{labeled precision: } = \frac{\# \text{ of correct constituents in hypothesis parse of } s}{\# \text{ of total constituents in hypothesis parse of } s}$$

F-measure As with other uses of precision and recall, we often report a combination of the two, the **F-measure** (van Rijsbergen, 1975), which, as we saw in Chapter 4, is defined as:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

Values of $\beta > 1$ favor recall and values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is called $F_{\beta=1}$ or just F_1 :

$$F_1 = \frac{2PR}{P+R} \quad (14.36)$$

We additionally use a new metric, crossing brackets, for each sentence s :

cross-brackets: the number of constituents for which the reference parse has a bracketing such as ((A B) C) but the hypothesis parse has a bracketing such as (A (B C)).

evalb For comparing parsers that use different grammars, the PARSEVAL metric includes a canonicalization algorithm for removing information likely to be grammar-specific (auxiliaries, pre-infinitival “to”, etc.) and for computing a simplified score (Black et al., 1991). The canonical implementation of the PARSEVAL metrics is called **evalb** (Sekine and Collins, 1997).

Nonetheless, phrasal constituents are not always an appropriate unit for parser evaluation. In lexically-oriented grammars, such as CCG and LFG, the ultimate goal is to extract the appropriate predicate-argument relations or grammatical dependencies, rather than a specific derivation. Such relations are also more directly relevant to further semantic processing. For these purposes, we can use alternative evaluation metrics based on the precision and recall of labeled dependencies whose labels indicate the grammatical relations (Lin 1995, Carroll et al. 1998, Collins et al. 1999).

Finally, you might wonder why we don’t evaluate parsers by measuring how many *sentences* are parsed correctly instead of measuring *component* accuracy in the form of constituents or dependencies. The reason we use components is that it gives us a more fine-grained metric. This is especially true for long sentences, where most parsers don’t get a perfect parse. If we just measured sentence accuracy, we wouldn’t be able to distinguish between a parse that got most of the parts wrong and one that just got one part wrong.

14.9 Summary

This chapter has sketched the basics of **probabilistic** parsing, concentrating on **probabilistic context-free grammars** and **probabilistic lexicalized context-free grammars**.

- Probabilistic grammars assign a probability to a sentence or string of words while attempting to capture more sophisticated syntactic information than the N -gram grammars of Chapter 3.

- A **probabilistic context-free grammar (PCFG)** is a context-free grammar in which every rule is annotated with the probability of that rule being chosen. Each PCFG rule is treated as if it were **conditionally independent**; thus, the probability of a sentence is computed by **multiplying** the probabilities of each rule in the parse of the sentence.
- The probabilistic CKY (**Cocke-Kasami-Younger**) algorithm is a probabilistic version of the CKY parsing algorithm. There are also probabilistic versions of other parsers like the Earley algorithm.
- PCFG probabilities can be learned by counting in a **parsed corpus** or by parsing a corpus. The **inside-outside** algorithm is a way of dealing with the fact that the sentences being parsed are ambiguous.
- Raw PCFGs suffer from poor independence assumptions among rules and lack of sensitivity to lexical dependencies.
- One way to deal with this problem is to split and merge non-terminals (automatically or by hand).
- **Probabilistic lexicalized CFGs** are another solution to this problem in which the basic PCFG model is augmented with a **lexical head** for each rule. The probability of a rule can then be conditioned on the lexical head or nearby heads.
- Parsers for lexicalized PCFGs (like the Charniak and Collins parsers) are based on extensions to probabilistic CKY parsing.
- Parsers are evaluated with three metrics: **labeled recall**, **labeled precision**, and **cross-brackets**.

Bibliographical and Historical Notes

Many of the formal properties of probabilistic context-free grammars were first worked out by [Booth \(1969\)](#) and [Salomaa \(1969\)](#). [Baker \(1979\)](#) proposed the inside-outside algorithm for unsupervised training of PCFG probabilities, and used a CKY-style parsing algorithm to compute inside probabilities. [Jelinek and Lafferty \(1991\)](#) extended the CKY algorithm to compute probabilities for prefixes. [Stolcke \(1995\)](#) adapted the Earley algorithm to use with PCFGs.

A number of researchers starting in the early 1990s worked on adding lexical dependencies to PCFGs and on making PCFG rule probabilities more sensitive to surrounding syntactic structure. For example, [Schabes et al. \(1988\)](#) and [Schabes \(1990\)](#) presented early work on the use of heads. Many papers on the use of lexical dependencies were first presented at the DARPA Speech and Natural Language Workshop in June 1990. A paper by [Hindle and Rooth \(1990\)](#) applied lexical dependencies to the problem of attaching prepositional phrases; in the question session to a later paper, Ken Church suggested applying this method to full parsing ([Marcus, 1990](#)). Early work on such probabilistic CFG parsing augmented with probabilistic dependency information includes [Magerman and Marcus \(1991\)](#), [Black et al. \(1992\)](#), [Bod \(1993\)](#), and [Jelinek et al. \(1994\)](#), in addition to [Collins \(1996\)](#), [Charniak \(1997\)](#), and [Collins \(1999\)](#) discussed above. Other recent PCFG parsing models include [Klein and Manning \(2003a\)](#) and [Petrov et al. \(2006\)](#).

This early lexical probabilistic work led initially to work focused on solving specific parsing problems like preposition-phrase attachment by using methods in-

cluding transformation-based learning (TBL) (Brill and Resnik, 1994), maximum entropy (Ratnaparkhi et al., 1994), memory-based learning (Zavrel and Daelemans, 1997), log-linear models (Franz, 1997), decision trees that used semantic distance between heads (computed from WordNet) (Stetina and Nagao, 1997), and boosting (Abney et al., 1999). Another direction extended the lexical probabilistic parsing work to build probabilistic formulations of grammars other than PCFGs, such as probabilistic TAG grammar (Resnik 1992, Schabes 1992), based on the TAG grammars discussed in Chapter 12, probabilistic LR parsing (Briscoe and Carroll, 1993), and probabilistic link grammar (Lafferty et al., 1992). The supertagging approach we saw for CCG was developed for TAG grammars (Bangalore and Joshi 1999, Joshi and Srinivas 1994), based on the lexicalized TAG grammars of Schabes et al. (1988).

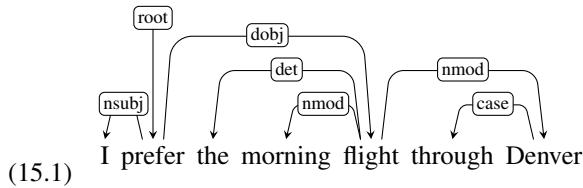
Exercises

- 14.1** Implement the CKY algorithm.
- 14.2** Modify the algorithm for conversion to CNF from Chapter 13 to correctly handle rule probabilities. Make sure that the resulting CNF assigns the same total probability to each parse tree.
- 14.3** Recall that Exercise 13.3 asked you to update the CKY algorithm to handle unit productions directly rather than converting them to CNF. Extend this change to probabilistic CKY.
- 14.4** Fill out the rest of the probabilistic CKY chart in Fig. 14.4.
- 14.5** Sketch how the CKY algorithm would have to be augmented to handle lexicalized probabilities.
- 14.6** Implement your lexicalized extension of the CKY algorithm.
- 14.7** Implement the PARSEVAL metrics described in Section 14.8. Next, either use a treebank or create your own hand-checked parsed test set. Now use your CFG (or other) parser and grammar, parse the test set and compute labeled recall, labeled precision, and cross-brackets.

dependency grammars

The focus of the three previous chapters has been on context-free grammars and their use in automatically generating constituent-based representations. Here we present another family of grammar formalisms called **dependency grammars** that are quite important in contemporary speech and language processing systems. In these formalisms, phrasal constituents and phrase-structure rules do not play a direct role. Instead, the syntactic structure of a sentence is described solely in terms of the words (or lemmas) in a sentence and an associated set of directed binary grammatical relations that hold among the words.

The following diagram illustrates a dependency-style analysis using the standard graphical method favored in the dependency-parsing community.



typed dependency

Relations among the words are illustrated above the sentence with directed, labeled arcs from heads to dependents. We call this a **typed dependency structure** because the labels are drawn from a fixed inventory of grammatical relations. It also includes a *root* node that explicitly marks the root of the tree, the head of the entire structure.

Figure 15.1 shows the same dependency analysis as a tree alongside its corresponding phrase-structure analysis of the kind given in Chapter 12. Note the absence of nodes corresponding to phrasal constituents or lexical categories in the dependency parse; the internal structure of the dependency parse consists solely of directed relations between lexical items in the sentence. These relationships directly encode important information that is often buried in the more complex phrase-structure parses. For example, the arguments to the verb *prefer* are directly linked to it in the dependency structure, while their connection to the main verb is more distant in the phrase-structure tree. Similarly, *morning* and *Denver*, modifiers of *flight*, are linked to it directly in the dependency structure.

free word order

A major advantage of dependency grammars is their ability to deal with languages that are morphologically rich and have a relatively **free word order**. For example, word order in Czech can be much more flexible than in English; a grammatical *object* might occur before or after a *location adverbial*. A phrase-structure grammar would need a separate rule for each possible place in the parse tree where such an adverbial phrase could occur. A dependency-based approach would just have one link type representing this particular adverbial relation. Thus, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse.

An additional practical motivation for a dependency-based approach is that the head-dependent relations provide an approximation to the semantic relationship be-



Figure 15.1 A dependency-style parse alongside the corresponding constituent-based analysis for *I prefer the morning flight through Denver.*

tween predicates and their arguments that makes them directly useful for many applications such as coreference resolution, question answering and information extraction. Constituent-based approaches to parsing provide similar information, but it often has to be distilled from the trees via techniques such as the head-finding rules discussed in Chapter 12.

In the following sections, we'll discuss in more detail the inventory of relations used in dependency parsing, as well as the formal basis for these dependency structures. We'll then move on to discuss the dominant families of algorithms that are used to automatically produce these structures. Finally, we'll discuss how to evaluate dependency parsers and point to some of the ways they are used in language processing applications.

15.1 Dependency Relations

grammatical relation The traditional linguistic notion of **grammatical relation** provides the basis for the binary relations that comprise these dependency structures. The arguments to these relations consist of a **head** and a **dependent**. We've already discussed the notion of heads in Chapter 12 and Chapter 14 in the context of constituent structures. There, the head word of a constituent was the central organizing word of a larger constituent (e.g. the primary noun in a noun phrase, or verb in a verb phrase). The remaining words in the constituent are either direct, or indirect, dependents of their head. In dependency-based approaches, the head-dependent relationship is made explicit by directly linking heads to the words that are immediately dependent on them, bypassing the need for constituent structures.

In addition to specifying the head-dependent pairs, dependency grammars allow us to further classify the kinds of grammatical relations, or **grammatical function**,

Clausal Argument Relations	Description
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
Nominal Modifier Relations	Description
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
Other Notable Relations	Description
CONJ	Conjunct
CC	Coordinating conjunction

Figure 15.2 Selected dependency relations from the Universal Dependency set. ([de Marneffe et al., 2014](#))

Universal
Dependencies

in terms of the role that the dependent plays with respect to its head. Familiar notions such as *subject*, *direct object* and *indirect object* are among the kind of relations we have in mind. In English these notions strongly correlate with, but by no means determine, both position in a sentence and constituent type and are therefore somewhat redundant with the kind of information found in phrase-structure trees. However, in more flexible languages the information encoded directly in these grammatical relations is critical since phrase-based constituent syntax provides little help.

Not surprisingly, linguists have developed taxonomies of relations that go well beyond the familiar notions of subject and object. While there is considerable variation from theory to theory, there is enough commonality that efforts to develop a computationally useful standard are now possible. The **Universal Dependencies** project ([Nivre et al., 2016b](#)) provides an inventory of dependency relations that are linguistically motivated, computationally useful, and cross-linguistically applicable. Fig. 15.2 shows a subset of the relations from this effort. Fig. 15.3 provides some example sentences illustrating selected relations.

The motivation for all of the relations in the Universal Dependency scheme is beyond the scope of this chapter, but the core set of frequently used relations can be broken into two sets: clausal relations that describe syntactic roles with respect to a predicate (often a verb), and modifier relations that categorize the ways that words that can modify their heads.

Consider the following example sentence:



The clausal relations NSUBJ and DOBJ identify the subject and direct object of the predicate *cancel*, while the NMOD, DET, and CASE relations denote modifiers of the nouns *flights* and *Houston*.

Relation	Examples with <i>head</i> and <i>dependent</i>
NSUBJ	United canceled the flight.
DOBJ	United diverted the flight to Reno.
IOBJ	We booked her the first flight to Miami.
NMOD	We booked her the flight to Miami.
AMOD	We took the morning flight.
NUMMOD	Before the storm JetBlue canceled 1000 flights.
APPOS	United, a unit of UAL, matched the fares.
DET	The flight was canceled.
CONJ	Which flight was delayed?
CC	We flew to Denver and drove to Steamboat.
CASE	We flew to Denver and drove to Steamboat.
	Book the flight through Houston.

Figure 15.3 Examples of core Universal Dependency relations.

15.2 Dependency Formalisms

In their most general form, the dependency structures we’re discussing are simply directed graphs. That is, structures $G = (V, A)$ consisting of a set of vertices V , and a set of ordered pairs of vertices A , which we’ll refer to as arcs.

For the most part we will assume that the set of vertices, V , corresponds exactly to the set of words in a given sentence. However, they might also correspond to punctuation, or when dealing with morphologically complex languages the set of vertices might consist of stems and affixes. The set of arcs, A , captures the head-dependent and grammatical function relationships between the elements in V .

Further constraints on these dependency structures are specific to the underlying grammatical theory or formalism. Among the more frequent restrictions are that the structures must be connected, have a designated root node, and be acyclic or planar. Of most relevance to the parsing approaches discussed in this chapter is the common, computationally-motivated, restriction to rooted trees. That is, a **dependency tree** is a directed graph that satisfies the following constraints:

1. There is a single designated root node that has no incoming arcs.
2. With the exception of the root node, each vertex has exactly one incoming arc.
3. There is a unique path from the root node to each vertex in V .

Taken together, these constraints ensure that each word has a single head, that the dependency structure is connected, and that there is a single root node from which one can follow a unique directed path to each of the words in the sentence.

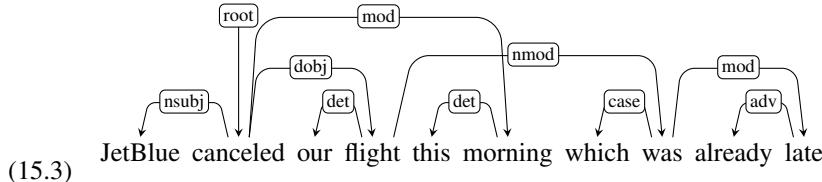
15.2.1 Projectivity

The notion of projectivity imposes an additional constraint that is derived from the order of the words in the input, and is closely related to the context-free nature of human languages discussed in Chapter 12. An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is then said to be projective if all the arcs that make it up are projective. All the dependency trees we’ve seen thus far have been projective. There are, however, many perfectly valid

dependency tree

constructions which lead to non-projective trees, particularly in languages with a relatively flexible word order.

Consider the following example.



In this example, the arc from *flight* to its modifier *was* is non-projective since there is no path from *flight* to the intervening words *this* and *morning*. As we can see from this diagram, projectivity (and non-projectivity) can be detected in the way we've been drawing our trees. A dependency tree is projective if it can be drawn with no crossing edges. Here there is no way to link *flight* to its dependent *was* without crossing the arc that links *morning* to its head.

Our concern with projectivity arises from two related issues. First, the most widely used English dependency treebanks were automatically derived from phrase-structure treebanks through the use of head-finding rules (Chapter 12). The trees generated in such a fashion are guaranteed to be projective since they're generated from context-free grammars.

Second, there are computational limitations to the most widely used families of parsing algorithms. The transition-based approaches discussed in Section 15.4 can only produce projective trees, hence any sentences with non-projective structures will necessarily contain some errors. This limitation is one of the motivations for the more flexible graph-based parsing approach described in Section 15.5.

15.3 Dependency Treebanks

As with constituent-based methods, treebanks play a critical role in the development and evaluation of dependency parsers. Dependency treebanks have been created using similar approaches to those discussed in Chapter 12 — having human annotators directly generate dependency structures for a given corpus, or using automatic parsers to provide an initial parse and then having annotators hand correct those parsers. We can also use a deterministic process to translate existing constituent-based treebanks into dependency trees through the use of head rules.

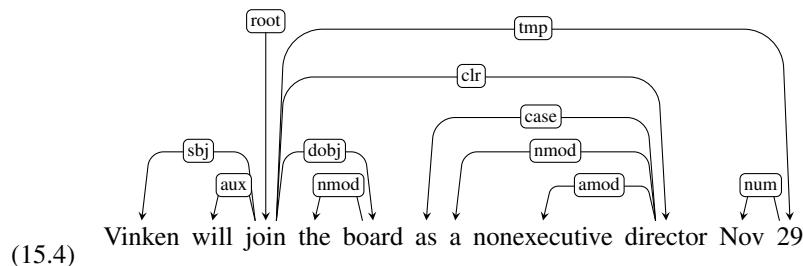
For the most part, directly annotated dependency treebanks have been created for morphologically rich languages such as Czech, Hindi and Finnish that lend themselves to dependency grammar approaches, with the Prague Dependency Treebank (Bejček et al., 2013) for Czech being the most well-known effort. The major English dependency treebanks have largely been extracted from existing resources such as the Wall Street Journal sections of the Penn Treebank (Marcus et al., 1993). The more recent OntoNotes project (Hovy et al. 2006, Weischedel et al. 2011) extends this approach going beyond traditional news text to include conversational telephone speech, weblogs, usenet newsgroups, broadcasts, and talk shows in English, Chinese and Arabic.

The translation process from constituent to dependency structures has two sub-tasks: identifying all the head-dependent relations in the structure and identifying the correct dependency relations for these relations. The first task relies heavily on

the use of head rules discussed in Chapter 12 first developed for use in lexicalized probabilistic parsers (Magerman 1994, Collins 1999, Collins 2003). Here's a simple and effective algorithm from Xia and Palmer (2001).

1. Mark the head child of each node in a phrase structure, using the appropriate head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

When a phrase-structure parse contains additional information in the form of grammatical relations and function tags, as in the case of the Penn Treebank, these tags can be used to label the edges in the resulting tree. When applied to the parse tree in Fig. 15.4, this algorithm would produce the dependency structure in example 15.4.



The primary shortcoming of these extraction methods is that they are limited by the information present in the original constituent trees. Among the most important issues are the failure to integrate morphological information with the phrase-structure trees, the inability to easily represent non-projective structures, and the lack of internal structure to most noun-phrases, as reflected in the generally flat rules used in most treebank grammars. For these reasons, outside of English, most dependency treebanks are developed directly using human annotators.

15.4 Transition-Based Dependency Parsing

shift-reduce parsing

Our first approach to dependency parsing is motivated by a stack-based approach called **shift-reduce parsing** originally developed for analyzing programming languages (Aho and Ullman, 1972). This classic approach is simple and elegant, employing a context-free grammar, a stack, and a list of tokens to be parsed. Input tokens are successively shifted onto the stack and the top two elements of the stack are matched against the right-hand side of the rules in the grammar; when a match is found the matched elements are replaced on the stack (reduced) by the non-terminal from the left-hand side of the rule being matched. In adapting this approach for dependency parsing, we forgo the explicit use of a grammar and alter the reduce operation so that instead of adding a non-terminal to a parse tree, it introduces a dependency relation between a word and its head. More specifically, the reduce action is replaced with two possible actions: assert a head-dependent relation between the word at the top of the stack and the word below it, or vice versa. Figure 15.5 illustrates the basic operation of such a parser.

configuration

A key element in transition-based parsing is the notion of a **configuration** which consists of a stack, an input buffer of words, or tokens, and a set of relations representing a dependency tree. Given this framework, the parsing process consists of

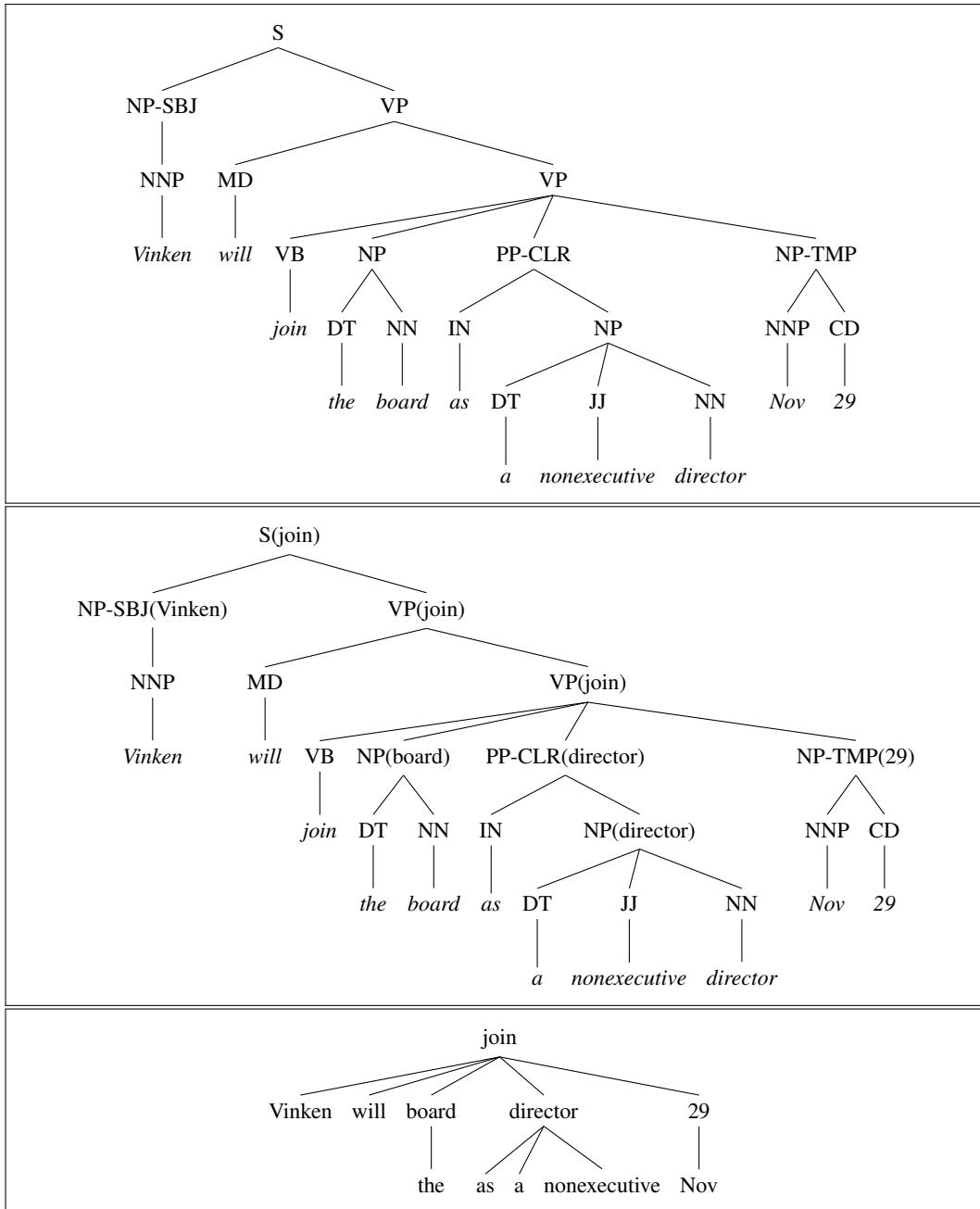


Figure 15.4 A phrase-structure tree from the *Wall Street Journal* component of the Penn Treebank 3.

a sequence of transitions through the space of possible configurations. The goal of this process is to find a final configuration where all the words have been accounted for and an appropriate dependency tree has been synthesized.

To implement such a search, we'll define a set of transition operators, which when applied to a configuration produce new configurations. Given this setup, we can view the operation of a parser as a search through a space of configurations for a sequence of transitions that leads from a start state to a desired goal state. At the start of this process we create an initial configuration in which the stack contains the



Figure 15.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

ROOT node, the word list is initialized with the set of the words or lemmatized tokens in the sentence, and an empty set of relations is created to represent the parse. In the final goal state, the stack and the word list should be empty, and the set of relations will represent the final parse.

In the standard approach to transition-based parsing, the operators used to produce new configurations are surprisingly simple and correspond to the intuitive actions one might take in creating a dependency tree by examining the words in a single pass over the input from left to right (Covington, 2001):

- Assign the current word as the head of some previously seen word,
- Assign some previously seen word as the head of the current word,
- Or postpone doing anything with the current word, adding it to a store for later processing.

To make these actions more precise, we'll create three transition operators that will operate on the top two elements of the stack:

- LEFTARC: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; remove the lower word from the stack.
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

arc standard

This particular set of operators implements what is known as the **arc standard** approach to transition-based parsing (Covington 2001, Nivre 2003). There are two notable characteristics to this approach: the transition operators only assert relations between elements at the top of the stack, and once an element has been assigned its head it is removed from the stack and is not available for further processing. As we'll see, there are alternative transition systems which demonstrate different parsing behaviors, but the arc standard approach is quite effective and is simple to implement.

To assure that these operators are used properly we'll need to add some preconditions to their use. First, since, by definition, the ROOT node cannot have any incoming arcs, we'll add the restriction that the LEFTARC operator cannot be applied when ROOT is the second element of the stack. Second, both reduce operators require two elements to be on the stack to be applied. Given these transition operators and preconditions, the specification of a transition-based parser is quite simple. Fig. 15.6 gives the basic algorithm.

```
function DEPENDENCYPARSE(words) returns dependency tree
    state  $\leftarrow \{[\text{root}], [\text{words}], []\}$  ; initial configuration
    while state not final
        t  $\leftarrow \text{ORACLE}(\text{state})$  ; choose a transition operator to apply
        state  $\leftarrow \text{APPLY}(t, \text{state})$  ; apply it, creating a new state
    return state
```

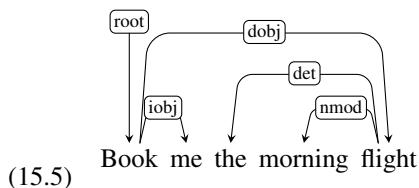
Figure 15.6 A generic transition-based dependency parser

At each step, the parser consults an oracle (we'll come back to this shortly) that provides the correct transition operator to use given the current configuration. It then applies that operator to the current configuration, producing a new configuration. The process ends when all the words in the sentence have been consumed and the ROOT node is the only element remaining on the stack.

The efficiency of transition-based parsers should be apparent from the algorithm. The complexity is linear in the length of the sentence since it is based on a single left to right pass through the words in the sentence. More specifically, each word must first be shifted onto the stack and then later reduced.

Note that unlike the dynamic programming and search-based approaches discussed in Chapters 12 and 13, this approach is a straightforward greedy algorithm — the oracle provides a single choice at each step and the parser proceeds with that choice, no other options are explored, no backtracking is employed, and a single parse is returned in the end.

Figure 15.7 illustrates the operation of the parser with the sequence of transitions leading to a parse for the following example.



Let's consider the state of the configuration at Step 2, after the word *me* has been pushed onto the stack.

Stack	Word List	Relations
[root, book, me]	[the, morning, flight]	

The correct operator to apply here is RIGHTARC which assigns *book* as the head of *me* and pops *me* from the stack resulting in the following configuration.

Stack	Word List	Relations
[root, book]	[the, morning, flight]	(book → me)

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

Figure 15.7 Trace of a transition-based parse.

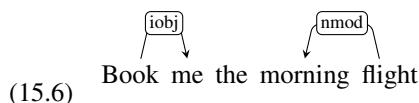
After several subsequent applications of the SHIFT and LEFTARC operators, the configuration in Step 6 looks like the following:

Stack	Word List	Relations
[root, book, the, morning, flight]	[]	(book → me)

Here, all the remaining words have been passed onto the stack and all that is left to do is to apply the appropriate reduce operators. In the current configuration, we employ the LEFTARC operator resulting in the following state.

Stack	Word List	Relations
[root, book, the, flight]	[]	(book → me) (morning ← flight)

At this point, the parse for this sentence consists of the following structure.



There are several important things to note when examining sequences such as the one in Figure 15.7. First, the sequence given is not the only one that might lead to a reasonable parse. In general, there may be more than one path that leads to the same result, and due to ambiguity, there may be other transition sequences that lead to different equally valid parses.

Second, we are assuming that the oracle always provides the correct operator at each point in the parse — an assumption that is unlikely to be true in practice. As a result, given the greedy nature of this algorithm, incorrect choices will lead to incorrect parses since the parser has no opportunity to go back and pursue alternative choices. Section 15.4.2 will introduce several techniques that allow transition-based approaches to explore the search space more fully.

Finally, for simplicity, we have illustrated this example without the labels on the dependency relations. To produce labeled trees, we can parameterize the LEFTARC and RIGHTARC operators with dependency labels, as in LEFTARC(NSUBJ) or RIGHTARC(DOBJ). This is equivalent to expanding the set of transition operators from our original set of three to a set that includes LEFTARC and RIGHTARC operators for each relation in the set of dependency relations being used, plus an additional one for the SHIFT operator. This, of course, makes the job of the oracle more difficult since it now has a much larger set of operators from which to choose.

15.4.1 Creating an Oracle

State-of-the-art transition-based systems use supervised machine learning methods to train classifiers that play the role of the oracle. Given appropriate training data, these methods learn a function that maps from configurations to transition operators.

As with all supervised machine learning methods, we will need access to appropriate training data and we will need to extract features useful for characterizing the decisions to be made. The source for this training data will be representative treebanks containing dependency trees. The features will consist of many of the same features we encountered in Chapter 8 for part-of-speech tagging, as well as those used in Chapter 14 for statistical parsing models.

Generating Training Data

Let's revisit the oracle from the algorithm in Fig. 15.6 to fully understand the learning problem. The oracle takes as input a configuration and returns as output a transition operator. Therefore, to train a classifier, we will need configurations paired with transition operators (i.e., LEFTARC, RIGHTARC, or SHIFT). Unfortunately, treebanks pair entire sentences with their corresponding trees, and therefore they don't directly provide what we need.

To generate the required training data, we will employ the oracle-based parsing algorithm in a clever way. We will supply our oracle with the training sentences to be parsed *along with* their corresponding reference parses from the treebank. To produce training instances, we will then *simulate* the operation of the parser by running the algorithm and relying on a new **training oracle** to give us correct transition operators for each successive configuration.

To see how this works, let's first review the operation of our parser. It begins with a default initial configuration where the stack contains the ROOT, the input list is just the list of words, and the set of relations is empty. The LEFTARC and RIGHTARC operators each add relations between the words at the top of the stack to the set of relations being accumulated for a given sentence. Since we have a gold-standard reference parse for each training sentence, we know which dependency relations are valid for a given sentence. Therefore, we can use the reference parse to guide the selection of operators as the parser steps through a sequence of configurations.

To be more precise, given a reference parse and a configuration, the training oracle proceeds as follows:

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration,
- Otherwise, choose RIGHTARC if (1) it produces a correct head-dependent relation given the reference parse and (2) all of the dependents of the word at the top of the stack have already been assigned,
- Otherwise, choose SHIFT.

The restriction on selecting the RIGHTARC operator is needed to ensure that a word is not popped from the stack, and thus lost to further processing, before all its dependents have been assigned to it.

More formally, during training the oracle has access to the following information:

- A current configuration with a stack S and a set of dependency relations R_c
- A reference parse consisting of a set of vertices V and a set of dependency relations R_p

training oracle

Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

Figure 15.8 Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.

Given this information, the oracle chooses transitions as follows:

LEFTARC(r): **if** $(S_1 r S_2) \in R_p$

RIGHTARC(r): **if** $(S_2 r S_1) \in R_p$ **and** $\forall r', w \text{ s.t. } (S_1 r' w) \in R_p$ **then** $(S_1 r' w) \in R_c$

SHIFT: **otherwise**

Let's walk through the steps of this process with the following example as shown in Fig. 15.8.



At Step 1, LEFTARC is not applicable in the initial configuration since it asserts a relation, $(\text{root} \leftarrow \text{book})$, not in the reference answer; RIGHTARC does assert a relation contained in the final answer $(\text{root} \rightarrow \text{book})$, however *book* has not been attached to any of its dependents yet, so we have to defer, leaving SHIFT as the only possible action. The same conditions hold in the next two steps. In step 3, LEFTARC is selected to link *the* to its head.

Now consider the situation in Step 4.

Stack	Word buffer	Relations
[root, book, flight]	[through, Houston]	(the \leftarrow flight)

Here, we might be tempted to add a dependency relation between *book* and *flight*, which is present in the reference parse. But doing so now would prevent the later attachment of *Houston* since *flight* would have been removed from the stack. Fortunately, the precondition on choosing RIGHTARC prevents this choice and we're again left with SHIFT as the only viable option. The remaining choices complete the set of operators needed for this example.

To recap, we derive appropriate training instances consisting of configuration-transition pairs from a treebank by simulating the operation of a parser in the context of a reference dependency tree. We can deterministically record correct parser actions at each step as we progress through each training example, thereby creating the training set we require.

Features

Having generated appropriate training instances (configuration-transition pairs), we need to extract useful features from the configurations so we can train classifiers. The features that are used to train transition-based systems vary by language, genre, and the kind of classifier being employed. For example, morphosyntactic features such as case marking on subjects or direct objects may be more or less important depending on the language being processed. That said, the basic features that we have already seen with part-of-speech tagging and partial parsing have proven to be useful in training dependency parsers across a wide range of languages. Word forms, lemmas and parts of speech are all powerful features, as are the head, and dependency relation to the head.

In the transition-based parsing framework, such features need to be extracted from the configurations that make up the training data. Recall that configurations consist of three elements: the stack, the buffer and the current set of relations. In principle, any property of any or all of these elements can be represented as features in the usual way for training. However, to avoid sparsity and encourage generalization, it is best to focus the learning algorithm on the most useful aspects of decision making at each point in the parsing process. The focus of feature extraction for transition-based parsing is, therefore, on the top levels of the stack, the words near the front of the buffer, and the dependency relations already associated with any of those elements.

feature template

By combining simple features, such as word forms or parts of speech, with specific locations in a configuration, we can employ the notion of a **feature template** that we've already encountered with sentiment analysis and part-of-speech tagging. Feature templates allow us to automatically generate large numbers of specific features from a training set. As an example, consider the following feature templates that are based on single positions in a configuration.

$$\begin{aligned} & \langle s_1.w, op \rangle, \langle s_2.w, op \rangle \langle s_1.t, op \rangle, \langle s_2.t, op \rangle \\ & \langle b_1.w, op \rangle, \langle b_1.t, op \rangle \langle s_1.wt, op \rangle \end{aligned} \quad (15.8)$$

In these examples, individual features are denoted as *location.property*, where *s* denotes the stack, *b* the word buffer, and *r* the set of relations. Individual properties of locations include *w* for word forms, *l* for lemmas, and *t* for part-of-speech. For example, the feature corresponding to the word form at the top of the stack would be denoted as $s_1.w$, and the part of speech tag at the front of the buffer $b_1.t$. We can also combine individual features via concatenation into more specific features that may prove useful. For example, the feature designated by $s_1.wt$ represents the word form concatenated with the part of speech of the word at the top of the stack. Finally, *op* stands for the transition operator for the training example in question (i.e., the label for the training instance).

Let's consider the simple set of single-element feature templates given above in the context of the following intermediate configuration derived from a training oracle for Example 15.2.

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

The correct transition here is SHIFT (you should convince yourself of this before

proceeding). The application of our set of feature templates to this configuration would result in the following set of instantiated features.

$$\begin{aligned}
 & \langle s_1.w = flights, op = shift \rangle & (15.9) \\
 & \langle s_2.w = canceled, op = shift \rangle \\
 & \langle s_1.t = NNS, op = shift \rangle \\
 & \langle s_2.t = VBD, op = shift \rangle \\
 & \langle b_1.w = to, op = shift \rangle \\
 & \langle b_1.t = TO, op = shift \rangle \\
 & \langle s_1.wt = flightsNNS, op = shift \rangle
 \end{aligned}$$

Given that the left and right arc transitions operate on the top two elements of the stack, features that *combine* properties from these positions are even more useful. For example, a feature like $s_1.t \circ s_2.t$ concatenates the part of speech tag of the word at the top of the stack with the tag of the word beneath it.

$$\langle s_1.t \circ s_2.t = NNSVBD, op = shift \rangle \quad (15.10)$$

Not surprisingly, if two properties are useful then three or more should be even better. Figure 15.9 gives a baseline set of feature templates that have been employed (Zhang and Clark 2008, Huang and Sagae 2010, Zhang and Nivre 2011).

Note that some of these features make use of *dynamic* features — features such as head words and dependency relations that have been predicted at earlier steps in the parsing process, as opposed to features that are derived from static properties of the input.

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Figure 15.9 Standard feature templates for training transition-based dependency parsers. In the template specifications s_n refers to a location on the stack, b_n refers to a location in the word buffer, w refers to the wordform of the input, and t refers to the part of speech of the input.

Learning

Over the years, the dominant approaches to training transition-based dependency parsers have been multinomial logistic regression and support vector machines, both of which can make effective use of large numbers of sparse features of the kind described in the last section. More recently, neural network, or deep learning, approaches of the kind described in Chapter 8 have been applied successfully to transition-based parsing (Chen and Manning, 2014). These approaches eliminate the need for complex, hand-crafted features and have been particularly effective at overcoming the data sparsity issues normally associated with training transition-based parsers.

15.4.2 Advanced Methods in Transition-Based Parsing

The basic transition-based approach can be elaborated in a number of ways to improve performance by addressing some of the most obvious flaws in the approach.

Alternative Transition Systems

The arc-standard transition system described above is only one of many possible systems. A frequently used alternative is the **arc eager** transition system. The arc eager approach gets its name from its ability to assert rightward relations much sooner than in the arc standard approach. To see this, let's revisit the arc standard trace of Example 15.7, repeated here.



Consider the dependency relation between *book* and *flight* in this analysis. As is shown in Fig. 15.8, an arc-standard approach would assert this relation at Step 8, despite the fact that *book* and *flight* first come together on the stack much earlier at Step 4. The reason this relation can't be captured at this point is due to the presence of the post-nominal modifier *through Houston*. In an arc-standard approach, dependents are removed from the stack as soon as they are assigned their heads. If *flight* had been assigned *book* as its head in Step 4, it would no longer be available to serve as the head of *Houston*.

While this delay doesn't cause any issues in this example, in general the longer a word has to wait to get assigned its head the more opportunities there are for something to go awry. The arc-eager system addresses this issue by allowing words to be attached to their heads as early as possible, before all the subsequent words dependent on them have been seen. This is accomplished through minor changes to the LEFTARC and RIGHTARC operators and the addition of a new REDUCE operator.

- LEFTARC: Assert a head-dependent relation between the word at the front of the input buffer and the word at the top of the stack; pop the stack.
- RIGHTARC: Assert a head-dependent relation between the word on the top of the stack and the word at front of the input buffer; shift the word at the front of the input buffer to the stack.
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.
- REDUCE: Pop the stack.

The LEFTARC and RIGHTARC operators are applied to the top of the stack and the front of the input buffer, instead of the top two elements of the stack as in the arc-standard approach. The RIGHTARC operator now moves the dependent to the stack from the buffer rather than removing it, thus making it available to serve as the head of following words. The new REDUCE operator removes the top element from the stack. Together these changes permit a word to be eagerly assigned its head and still allow it to serve as the head for later dependents. The trace shown in Fig. 15.10 illustrates the new decision sequence for this example.

In addition to demonstrating the arc-eager transition system, this example demonstrates the power and flexibility of the overall transition-based approach. We were able to swap in a new transition system without having to make any changes to the

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

Figure 15.10 A processing trace of *Book the flight through Houston* using the arc-eager transition operators.

underlying parsing algorithm. This flexibility has led to the development of a diverse set of transition systems that address different aspects of syntax and semantics including: assigning part of speech tags (Choi and Palmer, 2011a), allowing the generation of non-projective dependency structures (Nivre, 2009), assigning semantic roles (Choi and Palmer, 2011b), and parsing texts containing multiple languages (Bhat et al., 2017).

Beam Search

The computational efficiency of the transition-based approach discussed earlier derives from the fact that it makes a single pass through the sentence, greedily making decisions without considering alternatives. Of course, this is also the source of its greatest weakness – once a decision has been made it can not be undone, even in the face of overwhelming evidence arriving later in a sentence. Another approach is to systematically explore alternative decision sequences, selecting the best among those alternatives. The key problem for such a search is to manage the large number of potential sequences. **Beam search** accomplishes this by combining a breadth-first search strategy with a heuristic filter that prunes the search frontier to stay within a fixed-size **beam width**.

Beam search

beam width

In applying beam search to transition-based parsing, we’ll elaborate on the algorithm given in Fig. 15.6. Instead of choosing the single best transition operator at each iteration, we’ll apply all applicable operators to each state on an agenda and then score the resulting configurations. We then add each of these new configurations to the frontier, subject to the constraint that there has to be room within the beam. As long as the size of the agenda is within the specified beam width, we can add new configurations to the agenda. Once the agenda reaches the limit, we only add new configurations that are better than the worst configuration on the agenda (removing the worst element so that we stay within the limit). Finally, to insure that we retrieve the best possible state on the agenda, the while loop continues as long as there are non-final states on the agenda.

The beam search approach requires a more elaborate notion of scoring than we used with the greedy algorithm. There, we assumed that a classifier trained using supervised machine learning would serve as an oracle, selecting the best transition operator based on features extracted from the current configuration. Regardless of the specific learning approach, this choice can be viewed as assigning a score to all the possible transitions and picking the best one.

$$\hat{T}(c) = \text{argmaxScore}(t, c)$$

With a beam search we are now searching through the space of decision sequences, so it makes sense to base the score for a configuration on its entire history. More specifically, we can define the score for a new configuration as the score of its predecessor plus the score of the operator used to produce it.

$$\begin{aligned} \text{ConfigScore}(c_0) &= 0.0 \\ \text{ConfigScore}(c_i) &= \text{ConfigScore}(c_{i-1}) + \text{Score}(t_i, c_{i-1}) \end{aligned}$$

This score is used both in filtering the agenda and in selecting the final answer. The new beam search version of transition-based parsing is given in Fig. 15.11.

```

function DEPENDENCYBEAMPARSE(words, width) returns dependency tree
    state  $\leftarrow \{[\text{root}], [\text{words}], [], 0.0\}$  ;initial configuration
    agenda  $\leftarrow \langle \text{state} \rangle$  ;initial agenda

    while agenda contains non-final states
        newagenda  $\leftarrow \langle \rangle$ 
        for each state  $\in$  agenda do
            for all  $\{t \mid t \in \text{VALIDOPERATORS}(\text{state})\}$  do
                child  $\leftarrow \text{APPLY}(t, \text{state})
                newagenda  $\leftarrow \text{ADDTOBEAM}(\text{child}, \text{newagenda}, \text{width})$ 
            agenda  $\leftarrow \text{newagenda}$ 
        return  $\text{BESTOF}(\text{agenda})$ 

function ADDTOBEAM(state, agenda, width) returns updated agenda
    if LENGTH(agenda)  $<$  width then
        agenda  $\leftarrow \text{INSERT}(\text{state}, \text{agenda})$ 
    else if SCORE(state)  $>$  SCORE(WORSTOF(agenda))
        agenda  $\leftarrow \text{REMOVE}(\text{WORSTOF}(\text{agenda}))$ 
        agenda  $\leftarrow \text{INSERT}(\text{state}, \text{agenda})$ 
    return agenda$ 
```

Figure 15.11 Beam search applied to transition-based dependency parsing.

15.5 Graph-Based Dependency Parsing

Graph-based approaches to dependency parsing search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score. These methods encode the search space as directed graphs and employ methods drawn from graph theory to search the space for optimal solutions. More formally, given a sentence S we're looking for the best dependency tree in \mathcal{G}_S , the space of all possible trees for that sentence, that maximizes some score.

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}} \text{score}(t, S)$$

As with the probabilistic approaches to context-free parsing discussed in Chapter 14, the overall score for a tree can be viewed as a function of the scores of the parts of the tree. The focus of this section is on **edge-factored** approaches where the

score for a tree is based on the scores of the edges that comprise the tree.

$$\text{score}(t, S) = \sum_{e \in t} \text{score}(e)$$

There are several motivations for the use of graph-based methods. First, unlike transition-based approaches, these methods are capable of producing non-projective trees. Although projectivity is not a significant issue for English, it is definitely a problem for many of the world’s languages. A second motivation concerns parsing accuracy, particularly with respect to longer dependencies. Empirically, transition-based methods have high accuracy on shorter dependency relations but accuracy declines significantly as the distance between the head and dependent increases (McDonald and Nivre, 2011). Graph-based methods avoid this difficulty by scoring entire trees, rather than relying on greedy local decisions.

maximum spanning tree

The following section examines a widely-studied approach based on the use of a **maximum spanning tree** (MST) algorithm for weighted, directed graphs. We then discuss features that are typically used to score trees, as well as the methods used to train the scoring models.

15.5.1 Parsing

The approach described here uses an efficient greedy algorithm to search for optimal spanning trees in directed graphs. Given an input sentence, it begins by constructing a fully-connected, weighted, directed graph where the vertices are the input words and the directed edges represent *all possible* head-dependent assignments. An additional ROOT node is included with outgoing edges directed at all of the other vertices. The weights in the graph reflect the score for each possible head-dependent relation as provided by a model generated from training data. Given these weights, a maximum spanning tree of this graph emanating from the ROOT represents the preferred dependency parse for the sentence. A directed graph for the example *Book that flight* is shown in Fig. 15.12, with the maximum spanning tree corresponding to the desired parse shown in blue. For ease of exposition, we’ll focus here on unlabeled dependency parsing. Graph-based approaches to labeled parsing are discussed in Section 15.5.3.

Before describing the algorithm it’s useful to consider two intuitions about directed graphs and their spanning trees. The first intuition begins with the fact that every vertex in a spanning tree has exactly one incoming edge. It follows from this that every *connected component* of a spanning tree will also have one incoming edge. The second intuition is that the absolute values of the edge scores are not critical to determining its maximum spanning tree. Instead, it is the relative weights of the edges entering each vertex that matters. If we were to subtract a constant amount from each edge entering a given vertex it would have no impact on the choice of the maximum spanning tree since every possible spanning tree would decrease by exactly the same amount.

The first step of the algorithm itself is quite straightforward. For each vertex in the graph, an incoming edge (representing a possible head assignment) with the highest score is chosen. If the resulting set of edges produces a spanning tree then we’re done. More formally, given the original fully-connected graph $G = (V, E)$, a subgraph $T = (V, F)$ is a spanning tree if it has no cycles and each vertex (other than the root) has exactly one edge entering it. If the greedy selection process produces such a tree then it is the best possible one.

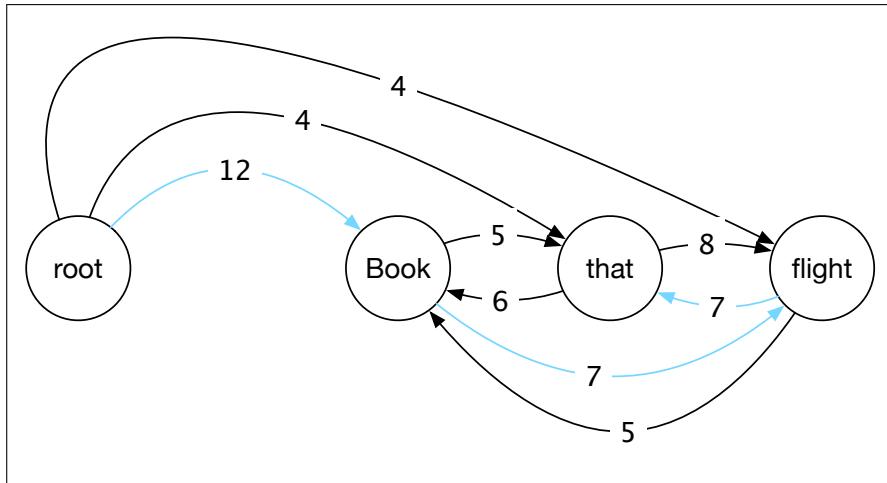


Figure 15.12 Initial rooted, directed graph for *Book that flight*.

Unfortunately, this approach doesn't always lead to a tree since the set of edges selected may contain cycles. Fortunately, in yet another case of multiple discovery, there is a straightforward way to eliminate cycles generated during the greedy selection phase. [Chu and Liu \(1965\)](#) and [Edmonds \(1967\)](#) independently developed an approach that begins with greedy selection and follows with an elegant recursive cleanup phase that eliminates cycles.

The cleanup phase begins by adjusting all the weights in the graph by subtracting the score of the maximum edge entering each vertex from the score of all the edges entering that vertex. This is where the intuitions mentioned earlier come into play. We have scaled the values of the edges so that the weight of the edges in the cycle have no bearing on the weight of *any* of the possible spanning trees. Subtracting the value of the edge with maximum weight from each edge entering a vertex results in a weight of zero for all of the edges selected during the greedy selection phase, *including all of the edges involved in the cycle*.

Having adjusted the weights, the algorithm creates a new graph by selecting a cycle and collapsing it into a single new node. Edges that enter or leave the cycle are altered so that they now enter or leave the newly collapsed node. Edges that do not touch the cycle are included and edges within the cycle are dropped.

Now, if we knew the maximum spanning tree of this new graph, we would have what we need to eliminate the cycle. The edge of the maximum spanning tree directed towards the vertex representing the collapsed cycle tells us which edge to delete to eliminate the cycle. How do we find the maximum spanning tree of this new graph? We recursively apply the algorithm to the new graph. This will either result in a spanning tree or a graph with a cycle. The recursions can continue as long as cycles are encountered. When each recursion completes we expand the collapsed vertex, restoring all the vertices and edges from the cycle *with the exception of the single edge to be deleted*.

Putting all this together, the maximum spanning tree algorithm consists of greedy edge selection, re-scoring of edge costs and a recursive cleanup phase when needed. The full algorithm is shown in Fig. 15.13.

Fig. 15.14 steps through the algorithm with our *Book that flight* example. The first row of the figure illustrates greedy edge selection with the edges chosen shown in blue (corresponding to the set F in the algorithm). This results in a cycle between

```

function MAXSPANNINGTREE( $G=(V,E)$ ,  $root$ ,  $score$ ) returns spanning tree
   $F \leftarrow []$ 
   $T' \leftarrow []$ 
   $score' \leftarrow []$ 
  for each  $v \in V$  do
     $bestInEdge \leftarrow \text{argmax}_{e=(u,v) \in E} score[e]$ 
     $F \leftarrow F \cup bestInEdge$ 
    for each  $e=(u,v) \in E$  do
       $score'[e] \leftarrow score[e] - score[bestInEdge]$ 

    if  $T=(V,F)$  is a spanning tree then return it
    else
       $C \leftarrow$  a cycle in  $F$ 
       $G' \leftarrow \text{CONTRACT}(G, C)$ 
       $T' \leftarrow \text{MAXSPANNINGTREE}(G', root, score')$ 
       $T \leftarrow \text{EXPAND}(T', C)$ 
      return  $T$ 

function CONTRACT( $G, C$ ) returns contracted graph
function EXPAND( $T, C$ ) returns expanded graph

```

Figure 15.13 The Chu-Liu Edmonds algorithm for finding a maximum spanning tree in a weighted directed graph.

that and *flight*. The scaled weights using the maximum value entering each node are shown in the graph to the right.

Collapsing the cycle between *that* and *flight* to a single node (labelled *tf*) and recursing with the newly scaled costs is shown in the second row. The greedy selection step in this recursion yields a spanning tree that links *root* to *book*, as well as an edge that links *book* to the contracted node. Expanding the contracted node, we can see that this edge corresponds to the edge from *book* to *flight* in the original graph. This in turn tells us which edge to drop to eliminate the cycle

On arbitrary directed graphs, this version of the CLE algorithm runs in $O(mn)$ time, where m is the number of edges and n is the number of nodes. Since this particular application of the algorithm begins by constructing a fully connected graph $m = n^2$ yielding a running time of $O(n^3)$. [Gabow et al. \(1986\)](#) present a more efficient implementation with a running time of $O(m + n\log n)$.

15.5.2 Features and Training

Given a sentence, S , and a candidate tree, T , edge-factored parsing models reduce the score for the tree to a sum of the scores of the edges that comprise the tree.

$$score(S, T) = \sum_{e \in T} score(S, e)$$

Each edge score can, in turn, be reduced to a weighted sum of features extracted from it.

$$score(S, e) = \sum_{i=1}^N w_i f_i(S, e)$$



Figure 15.14 Chu-Liu-Edmonds graph-based example for *Book that flight*

Or more succinctly,

$$\text{score}(S, e) = w \cdot f$$

Given this formulation, we are faced with two problems in training our parser: identifying relevant features and finding the weights used to score those features.

The features used to train edge-factored models mirror those used in training transition-based parsers (as shown in Fig. 15.9). This is hardly surprising since in both cases we’re trying to capture information about the relationship between heads and their dependents in the context of a single relation. To summarize this earlier discussion, commonly used features include:

- Wordforms, lemmas, and parts of speech of the headword and its dependent.
- Corresponding features derived from the contexts before, after and between the words.
- Word embeddings.
- The dependency relation itself.
- The direction of the relation (to the right or left).
- The distance from the head to the dependent.

As with transition-based approaches, pre-selected combinations of these features are often used as well.

Given a set of features, our next problem is to learn a set of weights corresponding to each. Unlike many of the learning problems discussed in earlier chapters,

inference-based learning

here we are not training a model to associate training items with class labels, or parser actions. Instead, we seek to train a model that assigns higher scores to correct trees than to incorrect ones. An effective framework for problems like this is to use **inference-based learning** combined with the perceptron learning rule. In this framework, we parse a sentence (i.e., perform inference) from the training set using some initially random set of initial weights. If the resulting parse matches the corresponding tree in the training data, we do nothing to the weights. Otherwise, we find those features in the incorrect parse that are *not* present in the reference parse and we lower their weights by a small amount based on the learning rate. We do this incrementally for each sentence in our training data until the weights converge.

State-of-the-art algorithms in multilingual parsing are based on recurrent neural networks (RNNs) (Zeman et al. 2017, Dozat et al. 2017).

15.5.3 Advanced Issues in Graph-Based Parsing

15.6 Evaluation

As with phrase structure-based parsing, the evaluation of dependency parsers proceeds by measuring how well they work on a test-set. An obvious metric would be exact match (EM) — how many sentences are parsed correctly. This metric is quite pessimistic, with most sentences being marked wrong. Such measures are not fine-grained enough to guide the development process. Our metrics need to be sensitive enough to tell if actual improvements are being made.

For these reasons, the most common method for evaluating dependency parsers are labeled and unlabeled attachment accuracy. Labeled attachment refers to the proper assignment of a word to its head along with the correct dependency relation. Unlabeled attachment simply looks at the correctness of the assigned head, ignoring the dependency relation. Given a system output and a corresponding reference parse, accuracy is simply the percentage of words in an input that are assigned the correct head with the correct relation. These metrics are usually referred to as the labeled attachment score (LAS) and unlabeled attachment score (UAS). Finally, we can make use of a label accuracy score (LS), the percentage of tokens with correct labels, ignoring where the relations are coming from.

As an example, consider the reference parse and system parse for the following example shown in Fig. 15.15.

(15.11) Book me the flight through Houston.

The system correctly finds 4 of the 6 dependency relations present in the reference parse and receives an LAS of 2/3. However, one of the 2 incorrect relations found by the system holds between *book* and *flight*, which are in a head-dependent relation in the reference parse; the system therefore achieves a UAS of 5/6.

Beyond attachment scores, we may also be interested in how well a system is performing on a particular kind of dependency relation, for example NSUBJ, across a development corpus. Here we can make use of the notions of precision and recall introduced in Chapter 8, measuring the percentage of relations labeled NSUBJ by the system that were correct (precision), and the percentage of the NSUBJ relations present in the development set that were in fact discovered by the system (recall). We can employ a confusion matrix to keep track of how often each dependency type was confused for another.



Figure 15.15 Reference and system parses for *Book me the flight through Houston*, resulting in an LAS of 2/3 and an UAS of 5/6.

15.7 Summary

This chapter has introduced the concept of dependency grammars and dependency parsing. Here's a summary of the main points that we covered:

- In dependency-based approaches to syntax, the structure of a sentence is described in terms of a set of binary relations that hold between the words in a sentence. Larger notions of constituency are not directly encoded in dependency analyses.
- The relations in a dependency structure capture the head-dependent relationship among the words in a sentence.
- Dependency-based analysis provides information directly useful in further language processing tasks including information extraction, semantic parsing and question answering.
- Transition-based parsing systems employ a greedy stack-based algorithm to create dependency structures.
- Graph-based methods for creating dependency structures are based on the use of maximum spanning tree methods from graph theory.
- Both transition-based and graph-based approaches are developed using supervised machine learning techniques.
- Treebanks provide the data needed to train these systems. Dependency treebanks can be created directly by human annotators or via automatic transformation from phrase-structure treebanks.
- Evaluation of dependency parsers is based on labeled and unlabeled accuracy scores as measured against withheld development and test corpora.

Bibliographical and Historical Notes

The dependency-based approach to grammar is much older than the relatively recent phrase-structure or constituency grammars that have been the primary focus of both theoretical and computational linguistics for years. It has its roots in the ancient Greek and Indian linguistic traditions. Contemporary theories of dependency grammar all draw heavily on the work of [Tesnière \(1959\)](#). The most influential dependency grammar frameworks include Meaning-Text Theory (MTT) ([Mel'čuk, 1988](#)), Word Grammar ([Hudson, 1984](#)), Functional Generative Description (FDG) ([Sgall et al., 1986](#)). These frameworks differ along a number of dimensions including the degree and manner in which they deal with morphological, syntactic,

semantic and pragmatic factors, their use of multiple layers of representation, and the set of relations used to categorize dependency relations.

Automatic parsing using dependency grammars was first introduced into computational linguistics by early work on machine translation at the RAND Corporation led by David Hays. This work on dependency parsing closely paralleled work on constituent parsing and made explicit use of grammars to guide the parsing process. After this early period, computational work on dependency parsing remained intermittent over the following decades. Notable implementations of dependency parsers for English during this period include Link Grammar ([Sleator and Temperley, 1993](#)), Constraint Grammar ([Karlsson et al., 1995](#)), and MINIPAR ([Lin, 2003](#)).

Dependency parsing saw a major resurgence in the late 1990’s with the appearance of large dependency-based treebanks and the associated advent of data driven approaches described in this chapter. [Eisner \(1996\)](#) developed an efficient dynamic programming approach to dependency parsing based on bilexical grammars derived from the Penn Treebank. [Covington \(2001\)](#) introduced the deterministic word by word approach underlying current transition-based approaches. [Yamada and Matsumoto \(2003\)](#) and [Kudo and Matsumoto \(2002\)](#) introduced both the shift-reduce paradigm and the use of supervised machine learning in the form of support vector machines to dependency parsing.

[Nivre \(2003\)](#) defined the modern, deterministic, transition-based approach to dependency parsing. Subsequent work by Nivre and his colleagues formalized and analyzed the performance of numerous transition systems, training methods, and methods for dealing with non-projective language [Nivre and Scholz 2004](#), [Nivre 2006](#), [Nivre and Nilsson 2005](#), [Nivre et al. 2007](#), [Nivre 2007](#).

The graph-based maximum spanning tree approach to dependency parsing was introduced by [McDonald et al. 2005](#), [McDonald et al. 2005](#).

The earliest source of data for training and evaluating dependency English parsers came from the WSJ Penn Treebank ([Marcus et al., 1993](#)) described in Chapter 12. The use of head-finding rules developed for use with probabilistic parsing facilitated the automatic extraction of dependency parses from phrase-based ones ([Xia and Palmer, 2001](#)).

The long-running Prague Dependency Treebank project ([Hajič, 1998](#)) is the most significant effort to directly annotate a corpus with multiple layers of morphological, syntactic and semantic information. The current PDT 3.0 now contains over 1.5 M tokens ([Bejček et al., 2013](#)).

Universal Dependencies (UD) ([Nivre et al., 2016b](#)) is a project directed at creating a consistent framework for dependency treebank annotation across languages with the goal of advancing parser development across the world’s languages. Under the auspices of this effort, treebanks for over 30 languages have been annotated and made available in a single consistent format. The UD annotation scheme evolved out of several distinct efforts including Stanford dependencies ([de Marneffe et al. 2006](#), [de Marneffe and Manning 2008](#), [de Marneffe et al. 2014](#)), Google’s universal part-of-speech tags ([Petrov et al., 2012](#)), and the Interset interlingua for morphosyntactic tagsets ([Zeman, 2008](#)). Driven in part by the UD framework, dependency treebanks of a significant size and quality are now available in over 30 languages ([Nivre et al., 2016b](#)).

The Conference on Natural Language Learning (CoNLL) has conducted an influential series of shared tasks related to dependency parsing over the years ([Buchholz and Marsi 2006](#), [Nilsson et al. 2007](#), [Surdeanu et al. 2008a](#), [Hajič et al. 2009](#)). More recent evaluations have focused on parser robustness with respect to morpho-

logically rich languages ([Seddah et al., 2013](#)), and non-canonical language forms such as social media, texts, and spoken language ([Petrov and McDonald, 2012](#)). [Choi et al. \(2015\)](#) presents a performance analysis of 10 dependency parsers across a range of metrics, as well as DEPENDABLE, a robust parser evaluation tool.

Exercises

CHAPTER

16 Logical Representations of Sentence Meaning

ISHMAEL: *Surely all this is not without meaning.*
Herman Melville, *Moby Dick*

meaning representations

In this chapter we introduce the idea that the meaning of linguistic expressions can be captured in formal structures called **meaning representations**. Consider tasks that require some form of semantic processing, like learning to use a new piece of software by reading the manual, deciding what to order at a restaurant by reading a menu, or following a recipe. Accomplishing these tasks requires representations that link the linguistic elements to the necessary non-linguistic *knowledge of the world*. Reading a menu and deciding what to order, giving advice about where to go to dinner, following a recipe, and generating new recipes all require knowledge about food and its preparation, what people like to eat, and what restaurants are like. Learning to use a piece of software by reading a manual, or giving advice on using software, requires knowledge about the software and similar apps, computers, and users in general.

semantic parsing
computational semantics

In this chapter, we assume that linguistic expressions have meaning representations that are made up of the *same kind of stuff* that is used to represent this kind of everyday common-sense knowledge of the world. The process whereby such representations are created and assigned to linguistic inputs is called **semantic parsing** or **semantic analysis**, and the entire enterprise of designing meaning representations and associated semantic parsers is referred to as **computational semantics**.

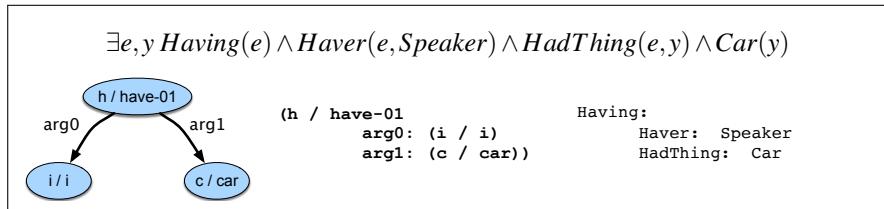


Figure 16.1 A list of symbols, two directed graphs, and a record structure: a sampler of meaning representations for *I have a car*.

Consider Fig. 16.1, which shows example meaning representations for the sentence *I have a car* using four commonly used meaning representation languages. The top row illustrates a sentence in **First-Order Logic**, covered in detail in Section 16.3; the directed graph and its corresponding textual form is an example of an **Abstract Meaning Representation (AMR)** form (Banerjee et al., 2013), and on the right is a **frame-based** or **slot-filler** representation, discussed in Section 16.5 and again in Chapter 18.

While there are non-trivial differences among these approaches, they all share the notion that a meaning representation consists of structures composed from a set of symbols, or representational vocabulary. When appropriately arranged, these symbol structures are taken to *correspond* to objects, properties of objects, and relations among objects in some state of affairs being represented or reasoned about. In this case, all four representations make use of symbols corresponding to the speaker, a car, and a relation denoting the possession of one by the other.

Importantly, these representations can be viewed from at least two distinct perspectives in all of these approaches: as representations of the meaning of the particular linguistic input *I have a car*, and as representations of the state of affairs in some world. It is this dual perspective that allows these representations to be used to link linguistic inputs to the world and to our knowledge of it.

In the next sections we give some background: our desiderata for a meaning representation language and some guarantees that these representations will actually do what we need them to do—provide a correspondence to the state of affairs being represented. In Section 16.3 we introduce First-Order Logic, historically the primary technique for investigating natural language semantics, and see in Section 16.4 how it can be used to capture the semantics of events and states in English. Chapter 17 then introduces techniques for **semantic parsing**: generating these formal meaning representations given linguistic inputs.

16.1 Computational Desiderata for Representations

Let's consider why meaning representations are needed and what they should do for us. To focus this discussion, let's consider a system that gives restaurant advice to tourists based on a knowledge base.

Verifiability

Consider the following simple question:

(16.1) Does Maharani serve vegetarian food?

To answer this question, we have to know what it's asking, and know whether what it's asking is true of Maharani or not. **verifiability** is a system's ability to compare the state of affairs described by a representation to the state of affairs in some world as modeled in a knowledge base. For example we'll need some sort of representation like *Serves(Maharani, VegetarianFood)*, which a system can match against its knowledge base of facts about particular restaurants, and if it finds a representation matching this proposition, it can answer yes. Otherwise, it must either say *No* if its knowledge of local restaurants is complete, or say that it doesn't know if it knows its knowledge is incomplete.

Unambiguous Representations

Semantics, like all the other domains we have studied, is subject to ambiguity. Words and sentences have different meaning representations in different contexts. Consider the following example:

(16.2) I wanna eat someplace that's close to ICSI.

This sentence can either mean that the speaker wants to eat *at* some nearby location, or under a Godzilla-as-speaker interpretation, the speaker may want to devour some

nearby location. The sentence is ambiguous; a single linguistic expression can have one of two meanings. But our *meaning representations* itself cannot be ambiguous. The representation of an input's meaning should be free from any ambiguity, so that the system can reason over a representation that means either one thing or the other in order to decide how to answer.

vagueness

A concept closely related to ambiguity is **vagueness**: in which a meaning representation leaves some parts of the meaning underspecified. Vagueness does not give rise to multiple representations. Consider the following request:

- (16.3) I want to eat Italian food.

While *Italian food* may provide enough information to provide recommendations, it is nevertheless *vague* as to what the user really wants to eat. A vague representation of the meaning of this phrase may be appropriate for some purposes, while a more specific representation may be needed for other purposes.

Canonical Form

canonical form

The doctrine of **canonical form** says that distinct inputs that mean the same thing should have the same meaning representation. This approach greatly simplifies reasoning, since systems need only deal with a single meaning representation for a potentially wide range of expressions.

Consider the following alternative ways of expressing (16.1):

- (16.4) Does Maharani have vegetarian dishes?
- (16.5) Do they have vegetarian food at Maharani?
- (16.6) Are vegetarian dishes served at Maharani?
- (16.7) Does Maharani serve vegetarian fare?

Despite the fact these alternatives use different words and syntax, we want them to map to a single canonical meaning representations. If they were all different, assuming the system's knowledge base contains only a single representation of this fact, most of the representations wouldn't match. We could, of course, store all possible alternative representations of the same fact in the knowledge base, but doing so would lead to enormous difficult in keeping the knowledge base consistent.

Canonical form does complicate the task of semantic parsing. Our system must conclude that *vegetarian fare*, *vegetarian dishes*, and *vegetarian food* refer to the same thing, that *having* and *serving* are equivalent here, and that all these parse structures still lead to the same meaning representation. Or consider this pair of examples:

- (16.8) Maharani serves vegetarian dishes.
- (16.9) Vegetarian dishes are served by Maharani.

Despite the different placement of the arguments to *serve*, a system must still assign *Maharani* and *vegetarian dishes* to the same roles in the two examples by drawing on grammatical knowledge, such as the relationship between active and passive sentence constructions.

Inference and Variables

What about more complex requests such as:

- (16.10) Can vegetarians eat at Maharani?

This request results in the same answer as the others not because they mean the same thing, but because there is a common-sense connection between what vegetarians eat

and what vegetarian restaurants serve. This is a fact about the world. We'll need to connect the meaning representation of this request with this fact about the world in a knowledge base. A system must be able to use **inference**—to draw valid conclusions based on the meaning representation of inputs and its background knowledge. It must be possible for the system to draw conclusions about the truth of propositions that are not explicitly represented in the knowledge base but that are nevertheless logically derivable from the propositions that are present.

Now consider the following somewhat more complex request:

(16.11) I'd like to find a restaurant where I can get vegetarian food.

This request does not make reference to any particular restaurant; the user wants information about an unknown restaurant that serves vegetarian food. Since no restaurants are named, simple matching is not going to work. Answering this request requires the use of **variables**, using some representation like the following:

$$\text{Serves}(x, \text{VegetarianFood}) \quad (16.12)$$

Matching succeeds only if the variable x can be replaced by some object in the knowledge base in such a way that the entire proposition will then match. The concept that is substituted for the variable can then be used to fulfill the user's request. It is critical for any meaning representation language to be able to handle these kinds of indefinite references.

Expressiveness

Finally, a meaning representation scheme must be expressive enough to handle a wide range of subject matter, ideally any sensible natural language utterance. Although this is probably too much to expect from any single representational system, First-Order Logic, as described in Section 16.3, is expressive enough to handle quite a lot of what needs to be represented.

16.2 Model-Theoretic Semantics

What is it about meaning representation languages that allows them to fulfill these desiderata, bridging the gap from formal representations to representations that tell us something about some state of affairs in the world?

The answer is a **model**. A model is a formal construct that stands for the particular state of affairs in the world. Expressions in a meaning representation language can be mapped to elements of the model, like objects, properties of objects, and relations among objects. If the model accurately captures the facts we're interested in, then a consistent mapping between the meaning representation and the model provides the bridge between meaning representation and world. Models provide a surprisingly simple and powerful way to ground the expressions in meaning representation languages.

First, some terminology. The vocabulary of a meaning representation consists of two parts: the non-logical vocabulary and the logical vocabulary. The **non-logical vocabulary** consists of the open-ended set of names for the objects, properties, and relations that make up the world we're trying to represent. These appear in various schemes as predicates, nodes, labels on links, or labels in slots in frames. The **logical vocabulary** consists of the closed set of symbols, operators, quantifiers, links,

non-logical vocabulary

logical vocabulary

etc., that provide the formal means for composing expressions in a given meaning representation language.

denotation

Each element of the non-logical vocabulary must have a **denotation** in the model, meaning that every element corresponds to a fixed, well-defined part of the model. Let's start with objects. The **domain** of a model is the set of objects that are being represented. Each distinct concept, category, or individual denotes a unique element in the domain.

extensional

We represent *properties* of objects in a model by denoting the domain elements that have the property; that is, properties denote sets. The denotation of the property *red* is the set of things we think are red. Similarly, a relations among object denote a set of ordered lists, or tuples, of domain elements that take part in the relations: the denotation of the relation *Married* is set of pairs of domain objects that are married. This approach to properties and relations is called **extensional**, because we define concepts by their extension, their denotations. To summarize:

- Objects denote *elements* of the domain
- Properties denote *sets of elements* of the domain
- Relations denote *sets of tuples of elements* of the domain

interpretation

We now need a mapping that gets us from our meaning representation to the corresponding denotations: a function that maps from the non-logical vocabulary of our meaning representation to the proper denotations in the model. We'll call such a mapping an **interpretation**.

Let's return to our restaurant advice application, and let its domain consist of sets of restaurants, patrons, facts about the likes and dislikes of the patrons, and facts about the restaurants such as their cuisine, typical cost, and noise level. To begin populating our domain, \emptyset , let's assume that we're dealing with four patrons designated by the non-logical symbols *Matthew*, *Franco*, *Katie*, and *Caroline*, denoting four unique domain elements. We'll use the constants a, b, c and, d to stand for these domain elements. We're deliberately using meaningless, non-mnemonic names for our domain elements to emphasize the fact that whatever it is that we know about these entities has to come from the formal properties of the model and not from the names of the symbols. Continuing, let's assume that our application includes three restaurants, designated as *Frasca*, *Med*, and *Rio* in our meaning representation, that denote the domain elements e, f , and g . Finally, let's assume that we're dealing with the three cuisines *Italian*, *Mexican*, and *Eclectic*, denoted by h, i , and j in our model.

Properties like *Noisy* denote the subset of restaurants from our domain that are known to be noisy. Two-place relational notions, such as which restaurants individual patrons *Like*, denote ordered pairs, or tuples, of the objects from the domain. And, since we decided to represent cuisines as objects in our model, we can capture which restaurants *Serve* which cuisines as a set of tuples. One possible state of affairs using this scheme is given in Fig. 16.2.

Given this simple scheme, we can ground our meaning representations by consulting the appropriate denotations in the corresponding model. For example, we can evaluate a representation claiming that *Matthew likes the Rio*, or that *The Med serves Italian* by mapping the objects in the meaning representations to their corresponding domain elements and mapping any links, predicates, or slots in the meaning representation to the appropriate relations in the model. More concretely, we can verify a representation asserting that *Matthew likes Frasca* by first using our interpretation function to map the symbol *Matthew* to its denotation a , *Frasca* to e , and the *Likes* relation to the appropriate set of tuples. We then check that set of tuples for the

Domain	$\mathcal{D} = \{a, b, c, d, e, f, g, h, i, j\}$
Matthew, Franco, Katie and Caroline	a, b, c, d
Frasca, Med, Rio	e, f, g
Italian, Mexican, Eclectic	h, i, j
Properties	
Noisy	$Noisy = \{e, f, g\}$
Frasca, Med, and Rio are noisy	
Relations	
Likes	$Likes = \{\langle a, f \rangle, \langle c, f \rangle, \langle c, g \rangle, \langle b, e \rangle, \langle d, f \rangle, \langle d, g \rangle\}$
Matthew likes the Med	
Katie likes the Med and Rio	
Franco likes Frasca	
Caroline likes the Med and Rio	
Serves	$Serves = \{\langle f, j \rangle, \langle g, i \rangle, \langle e, h \rangle\}$
Med serves eclectic	
Rio serves Mexican	
Frasca serves Italian	

Figure 16.2 A model of the restaurant world.

presence of the tuple $\langle a, e \rangle$. If, as it is in this case, the tuple is present in the model, then we can conclude that *Matthew likes Frasca* is true; if it isn't then we can't.

This is all pretty straightforward—we're using sets and operations on sets to ground the expressions in our meaning representations. Of course, the more interesting part comes when we consider more complex examples such as the following:

- (16.13) Katie likes the Rio and Matthew likes the Med.
- (16.14) Katie and Caroline like the same restaurants.
- (16.15) Franco likes noisy, expensive restaurants.
- (16.16) Not everybody likes Frasca.

Our simple scheme for grounding the meaning of representations is not adequate for examples such as these. Plausible meaning representations for these examples will not map directly to individual entities, properties, or relations. Instead, they involve complications such as conjunctions, equality, quantified variables, and negations. To assess whether these statements are consistent with our model, we'll have to tear them apart, assess the parts, and then determine the meaning of the whole from the meaning of the parts.

Consider the first example above. A meaning representation for this example will include two distinct propositions expressing the individual patron's preferences, conjoined with some kind of implicit or explicit conjunction operator. Our model doesn't have a relation that encodes pairwise preferences for all of the patrons and restaurants in our model, nor does it need to. We know from our model that *Matthew likes the Med* and separately that *Katie likes the Rio* (that is, the tuples $\langle a, f \rangle$ and $\langle c, g \rangle$ are members of the set denoted by the *Likes* relation). All we really need to know is how to deal with the semantics of the conjunction operator. If we assume the simplest possible semantics for the English word *and*, the whole statement is true if it is the case that each of the components is true in our model. In this case, both components are true since the appropriate tuples are present and therefore the sentence as a whole is true.

What we've done with this example is provide a **truth-conditional semantics**

<i>Formula</i>	\rightarrow	<i>AtomicFormula</i>
		<i>Formula Connective Formula</i>
		<i>Quantifier Variable, ... Formula</i>
		\neg <i>Formula</i>
		(<i>Formula</i>)
<i>AtomicFormula</i>	\rightarrow	<i>Predicate(Term, ...)</i>
<i>Term</i>	\rightarrow	<i>Function(Term, ...)</i>
		<i>Constant</i>
		<i>Variable</i>
<i>Connective</i>	\rightarrow	\wedge \vee \Rightarrow
<i>Quantifier</i>	\rightarrow	\forall \exists
<i>Constant</i>	\rightarrow	<i>A</i> <i>VegetarianFood</i> <i>Maharani</i> ...
<i>Variable</i>	\rightarrow	<i>x</i> <i>y</i> ...
<i>Predicate</i>	\rightarrow	<i>Serves</i> <i>Near</i> ...
<i>Function</i>	\rightarrow	<i>LocationOf</i> <i>CuisineOf</i> ...

Figure 16.3 A context-free grammar specification of the syntax of First-Order Logic representations. Adapted from Russell and Norvig (2002).

for the assumed conjunction operator in some meaning representation. That is, we've provided a method for determining the truth of a complex expression from the meanings of the parts (by consulting a model) and the meaning of an operator by consulting a truth table. Meaning representation languages are truth-conditional to the extent that they give a formal specification as to how we can determine the meaning of complex sentences from the meaning of their parts. In particular, we need to know the semantics of the entire logical vocabulary of the meaning representation scheme being used.

Note that although the details of how this happens depend on details of the particular meaning representation being used, it should be clear that assessing the truth conditions of examples like these involves nothing beyond the simple set operations we've been discussing. We return to these issues in the next section in the context of the semantics of First-Order Logic.

16.3 First-Order Logic

First-Order Logic (FOL) is a flexible, well-understood, and computationally tractable meaning representation language that satisfies many of the desiderata given in Section 16.1. It provides a sound computational basis for the verifiability, inference, and expressiveness requirements, as well as a sound model-theoretic semantics.

An additional attractive feature of FOL is that it makes few specific commitments as to how things ought to be represented, and those it does are shared by many of the schemes mentioned earlier: the represented world consists of objects, properties of objects, and relations among objects.

The remainder of this section introduces the basic syntax and semantics of FOL and then describes the application of FOL to the representation of events.

16.3.1 Basic Elements of First-Order Logic

Let's explore FOL by first examining its various atomic elements and then showing how they can be composed to create larger meaning representations. Figure 16.3,

which provides a complete context-free grammar for the particular syntax of FOL that we will use, is our roadmap for this section.

term

Let's begin by examining the notion of a **term**, the FOL device for representing objects. As can be seen from Fig. 16.3, FOL provides three ways to represent these basic building blocks: constants, functions, and variables. Each of these devices can be thought of as designating an object in the world under consideration.

constant

Constants in FOL refer to specific objects in the world being described. Such constants are conventionally depicted as either single capitalized letters such as *A* and *B* or single capitalized words that are often reminiscent of proper nouns such as *Maharani* and *Harry*. Like programming language constants, FOL constants refer to exactly one object. Objects can, however, have multiple constants that refer to them.

function

Functions in FOL correspond to concepts that are often expressed in English as genitives such as *Frasca's location*. A FOL translation of such an expression might look like the following.

$$\text{LocationOf}(\text{Frasca}) \quad (16.17)$$

FOL functions are syntactically the same as single argument predicates. It is important to remember, however, that while they have the appearance of predicates, they are in fact *terms* in that they refer to unique objects. Functions provide a convenient way to refer to specific objects without having to associate a named constant with them. This is particularly convenient in cases in which many named objects, like restaurants, have a unique concept such as a location associated with them.

variable

Variables are our final FOL mechanism for referring to objects. Variables, depicted as single lower-case letters, let us make assertions and draw inferences about objects without having to make reference to any particular named object. This ability to make statements about anonymous objects comes in two flavors: making statements about a particular unknown object and making statements about all the objects in some arbitrary world of objects. We return to the topic of variables after we have presented quantifiers, the elements of FOL that make variables useful.

Now that we have the means to refer to objects, we can move on to the FOL mechanisms that are used to state relations that hold among objects. Predicates are symbols that refer to, or name, the relations that hold among some fixed number of objects in a given domain. Returning to the example introduced informally in Section 16.1, a reasonable FOL representation for *Maharani serves vegetarian food* might look like the following formula:

$$\text{Serves}(\text{Maharani}, \text{VegetarianFood}) \quad (16.18)$$

This FOL sentence asserts that *Serves*, a two-place predicate, holds between the objects denoted by the constants *Maharani* and *VegetarianFood*.

A somewhat different use of predicates is illustrated by the following fairly typical representation for a sentence like *Maharani is a restaurant*:

$$\text{Restaurant}(\text{Maharani}) \quad (16.19)$$

This is an example of a one-place predicate that is used, not to relate multiple objects, but rather to assert a property of a single object. In this case, it encodes the category membership of *Maharani*.

With the ability to refer to objects, to assert facts about objects, and to relate objects to one another, we can create rudimentary composite representations. These representations correspond to the atomic formula level in Fig. 16.3. This ability

logical connectives

to compose complex representations is, however, not limited to the use of single predicates. Larger composite representations can also be put together through the use of **logical connectives**. As can be seen from Fig. 16.3, logical connectives let us create larger representations by conjoining logical formulas using one of three operators. Consider, for example, the following BERP sentence and one possible representation for it:

(16.20) I only have five dollars and I don't have a lot of time.

$$\text{Have}(\text{Speaker}, \text{FiveDollars}) \wedge \neg \text{Have}(\text{Speaker}, \text{LotOfTime}) \quad (16.21)$$

The semantic representation for this example is built up in a straightforward way from the semantics of the individual clauses through the use of the \wedge and \neg operators. Note that the recursive nature of the grammar in Fig. 16.3 allows an infinite number of logical formulas to be created through the use of these connectives. Thus, as with syntax, we can use a finite device to create an infinite number of representations.

quantifiers

16.3.2 Variables and Quantifiers

We now have all the machinery necessary to return to our earlier discussion of variables. As noted above, variables are used in two ways in FOL: to refer to particular anonymous objects and to refer generically to all objects in a collection. These two uses are made possible through the use of operators known as **quantifiers**. The two operators that are basic to FOL are the existential quantifier, which is denoted \exists and is pronounced as “there exists”, and the universal quantifier, which is denoted \forall and is pronounced as “for all”.

The need for an existentially quantified variable is often signaled by the presence of an indefinite noun phrase in English. Consider the following example:

(16.22) a restaurant that serves Mexican food near ICSI.

Here, reference is being made to an anonymous object of a specified category with particular properties. The following would be a reasonable representation of the meaning of such a phrase:

$$\begin{aligned} \exists x \text{Restaurant}(x) \wedge & \text{Serves}(x, \text{MexicanFood}) \\ \wedge & \text{Near}((\text{LocationOf}(x), \text{LocationOf}(\text{ICSI})) \end{aligned} \quad (16.23)$$

The existential quantifier at the head of this sentence instructs us on how to interpret the variable x in the context of this sentence. Informally, it says that for this sentence to be true there must be at least one object such that if we were to substitute it for the variable x , the resulting sentence would be true. For example, if *AyCaramba* is a Mexican restaurant near ICSI, then substituting *AyCaramba* for x results in the following logical formula:

$$\begin{aligned} \text{Restaurant}(\text{AyCaramba}) \wedge \text{Serves}(\text{AyCaramba}, \text{MexicanFood}) \\ \wedge \text{Near}((\text{LocationOf}(\text{AyCaramba}), \text{LocationOf}(\text{ICSI})) \end{aligned} \quad (16.24)$$

Based on the semantics of the \wedge operator, this sentence will be true if all of its three component atomic formulas are true. These in turn will be true if they are either present in the system’s knowledge base or can be inferred from other facts in the knowledge base.

The use of the universal quantifier also has an interpretation based on substitution of known objects for variables. The substitution semantics for the universal

quantifier takes the expression *for all* quite literally; the \forall operator states that for the logical formula in question to be true, the substitution of *any* object in the knowledge base for the universally quantified variable should result in a true formula. This is in marked contrast to the \exists operator, which only insists on a single valid substitution for the sentence to be true.

Consider the following example:

(16.25) All vegetarian restaurants serve vegetarian food.

A reasonable representation for this sentence would be something like the following:

$$\forall x \text{VegetarianRestaurant}(x) \implies \text{Serves}(x, \text{VegetarianFood}) \quad (16.26)$$

For this sentence to be true, every substitution of a known object for x must result in a sentence that is true. We can divide the set of all possible substitutions into the set of objects consisting of vegetarian restaurants and the set consisting of everything else. Let us first consider the case in which the substituted object actually is a vegetarian restaurant; one such substitution would result in the following sentence:

$$\text{VegetarianRestaurant}(\text{Maharani}) \implies \text{Serves}(\text{Maharani}, \text{VegetarianFood}) \quad (16.27)$$

If we assume that we know that the consequent clause

$$\text{Serves}(\text{Maharani}, \text{VegetarianFood}) \quad (16.28)$$

is true, then this sentence as a whole must be true. Both the antecedent and the consequent have the value *True* and, therefore, according to the first two rows of Fig. 16.4 on page 309 the sentence itself can have the value *True*. This result will be the same for all possible substitutions of *Terms* representing vegetarian restaurants for x .

Remember, however, that for this sentence to be true, it must be true for all possible substitutions. What happens when we consider a substitution from the set of objects that are not vegetarian restaurants? Consider the substitution of a non-vegetarian restaurant such as *AyCaramba* for the variable x :

$$\text{VegetarianRestaurant}(\text{AyCaramba}) \implies \text{Serves}(\text{AyCaramba}, \text{VegetarianFood})$$

Since the antecedent of the implication is *False*, we can determine from Fig. 16.4 that the sentence is always *True*, again satisfying the \forall constraint.

Note that it may still be the case that *AyCaramba* serves vegetarian food without actually being a vegetarian restaurant. Note also that, despite our choice of examples, there are no implied categorical restrictions on the objects that can be substituted for x by this kind of reasoning. In other words, there is no restriction of x to restaurants or concepts related to them. Consider the following substitution:

$$\text{VegetarianRestaurant}(\text{Carburetor}) \implies \text{Serves}(\text{Carburetor}, \text{VegetarianFood})$$

Here the antecedent is still false so the rule remains true under this kind of irrelevant substitution.

To review, variables in logical formulas must be either existentially (\exists) or universally (\forall) quantified. To satisfy an existentially quantified variable, at least one substitution must result in a true sentence. To satisfy a universally quantified variable, all substitutions must result in true sentences.

16.3.3 Lambda Notation

lambda notation

The final element we need to complete our discussion of FOL is called the **lambda notation** (Church, 1940). This notation provides a way to abstract from fully specified FOL formulas in a way that will be particularly useful for semantic analysis. The lambda notation extends the syntax of FOL to include expressions of the following form:

$$\lambda x.P(x) \quad (16.29)$$

 λ -reduction

Such expressions consist of the Greek symbol λ , followed by one or more variables, followed by a FOL formula that makes use of those variables.

The usefulness of these λ -expressions is based on the ability to apply them to logical terms to yield new FOL expressions where the formal parameter variables are bound to the specified terms. This process is known as **λ -reduction**, and consists of a simple textual replacement of the λ variables and the removal of the λ . The following expressions illustrate the application of a λ -expression to the constant A , followed by the result of performing a λ -reduction on this expression:

$$\begin{aligned} \lambda x.P(x)(A) \\ P(A) \end{aligned} \quad (16.30)$$

An important and useful variation of this technique is the use of one λ -expression as the body of another as in the following expression:

$$\lambda x.\lambda y.Near(x,y) \quad (16.31)$$

This fairly abstract expression can be glossed as the state of something being near something else. The following expressions illustrate a single λ -application and subsequent reduction with this kind of embedded λ -expression:

$$\begin{aligned} \lambda x.\lambda y.Near(x,y)(Bacaro) \\ \lambda y.Near(Bacaro,y) \end{aligned} \quad (16.32)$$

The important point here is that the resulting expression is still a λ -expression; the first reduction bound the variable x and removed the outer λ , thus revealing the inner expression. As might be expected, this resulting λ -expression can, in turn, be applied to another term to arrive at a fully specified logical formula, as in the following:

$$\begin{aligned} \lambda y.Near(Bacaro,y)(Centro) \\ Near(Bacaro,Centro) \end{aligned} \quad (16.33)$$

currying

This general technique, called **currying**¹ (Schönfinkel, 1924) is a way of converting a predicate with multiple arguments into a sequence of single-argument predicates.

As we show in Chapter 17, the λ -notation provides a way to incrementally gather arguments to a predicate when they do not all appear together as daughters of the predicate in a parse tree.

16.3.4 The Semantics of First-Order Logic

The various objects, properties, and relations represented in a FOL knowledge base acquire their meanings by virtue of their correspondence to objects, properties, and

¹ Currying is the standard term, although Heim and Kratzer (1998) present an interesting argument for the term *Schönfinkelization* over currying, since Curry later built on Schönfinkel's work.

relations out in the external world being modeled. We can accomplish this by employing the model-theoretic approach introduced in Section 16.2. Recall that this approach employs simple set-theoretic notions to provide a truth-conditional mapping from the expressions in a meaning representation to the state of affairs being modeled. We can apply this approach to FOL by going through all the elements in Fig. 16.3 on page 304 and specifying how each should be accounted for.

We can start by asserting that the objects in our world, FOL terms, denote elements in a domain, and asserting that atomic formulas are captured either as sets of domain elements for properties, or as sets of tuples of elements for relations. As an example, consider the following:

(16.34) Centro is near Bacaro.

Capturing the meaning of this example in FOL involves identifying the *Terms* and *Predicates* that correspond to the various grammatical elements in the sentence and creating logical formulas that capture the relations implied by the words and syntax of the sentence. For this example, such an effort might yield something like the following:

$$\text{Near}(\text{Centro}, \text{Bacaro}) \quad (16.35)$$

The meaning of this logical formula is based on whether the domain elements denoted by the terms *Centro* and *Bacaro* are contained among the tuples denoted by the relation denoted by the predicate *Near* in the current model.

The interpretation of formulas involving logical connectives is based on the meanings of the components in the formulas combined with the meanings of the connectives they contain. Figure 16.4 gives interpretations for each of the logical operators shown in Fig. 16.3.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$
<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>	<i>False</i>	<i>True</i>
<i>False</i>	<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>
<i>True</i>	<i>False</i>	<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>
<i>True</i>	<i>True</i>	<i>False</i>	<i>True</i>	<i>True</i>	<i>True</i>

Figure 16.4 Truth table giving the semantics of the various logical connectives.

The semantics of the \wedge (and) and \neg (not) operators are fairly straightforward, and are correlated with at least some of the senses of the corresponding English terms. However, it is worth pointing out that the \vee (or) operator is not disjunctive in the same way that the corresponding English word is, and that the \Rightarrow (implies) operator is only loosely based on any common-sense notions of implication or causation.

The final bit we need to address involves variables and quantifiers. Recall that there are no variables in our set-based models, only elements of the domain and relations that hold among them. We can provide a model-based account for formulas with variables by employing the notion of a substitution introduced earlier on page 306. Formulas involving \exists are true if a substitution of terms for variables results in a formula that is true in the model. Formulas involving \forall must be true under all possible substitutions.

16.3.5 Inference

A meaning representation language must support inference to add valid new propositions to a knowledge base or to determine the truth of propositions not explicitly

Modus ponens

contained within a knowledge base (Section 16.1). This section briefly discusses **modus ponens**, the most widely implemented inference method provided by FOL.

Modus ponens is a form of inference that corresponds to what is informally known as *if-then* reasoning. We can abstractly define modus ponens as follows, where α and β should be taken as FOL formulas:

$$\frac{\alpha}{\frac{\alpha \implies \beta}{\beta}} \quad (16.36)$$

A schema like this indicates that the formula below the line can be inferred from the formulas above the line by some form of inference. **Modus ponens** states that if the left-hand side of an implication rule is true, then the right-hand side of the rule can be inferred. In the following discussions, we will refer to the left-hand side of an implication as the **antecedent** and the right-hand side as the **consequent**.

For a typical use of modus ponens, consider the following example, which uses a rule from the last section:

$$\frac{\begin{array}{c} \text{VegetarianRestaurant(Leaf)} \\ \forall x \text{VegetarianRestaurant}(x) \implies \text{Serves}(x, \text{VegetarianFood}) \end{array}}{\text{Serves(Leaf, VegetarianFood)}} \quad (16.37)$$

Here, the formula *VegetarianRestaurant(Leaf)* matches the antecedent of the rule, thus allowing us to use modus ponens to conclude *Serves(Leaf, VegetarianFood)*.

forward chaining

Modus ponens can be put to practical use in one of two ways: forward chaining and backward chaining. In **forward chaining** systems, modus ponens is used in precisely the manner just described. As individual facts are added to the knowledge base, modus ponens is used to fire all applicable implication rules. In this kind of arrangement, as soon as a new fact is added to the knowledge base, all applicable implication rules are found and applied, each resulting in the addition of new facts to the knowledge base. These new propositions in turn can be used to fire implication rules applicable to them. The process continues until no further facts can be deduced.

The forward chaining approach has the **advantage** that facts will be present in the knowledge base when needed, because, in a sense all inference is performed in advance. This can substantially reduce the time needed to answer subsequent queries since they should all amount to simple lookups. The **disadvantage** of this approach is that facts that will never be needed may be inferred and stored.

backward chaining

In **backward chaining**, modus ponens is run in reverse to prove specific propositions called **queries**. The first step is to see if the query formula is true by determining if it is present in the knowledge base. If it is not, then the next step is to search for applicable implication rules present in the knowledge base. An applicable rule is one whereby the consequent of the rule matches the query formula. If there are any such rules, then the query can be proved if the antecedent of any one them can be shown to be true. This can be performed recursively by backward chaining on the antecedent as a new query. The Prolog programming language is a backward chaining system that implements this strategy.

To see how this works, let's assume that we have been asked to verify the truth of the proposition *Serves(Leaf, VegetarianFood)*, assuming the facts given above the line in (16.37). Since this proposition is not present in the knowledge base, a search for an applicable rule is initiated resulting in the rule given above. After substituting

the constant *Leaf* for the variable *x*, our next task is to prove the antecedent of the rule, *VegetarianRestaurant(Leaf)*, which, of course, is one of the facts we are given.

Note that it is critical to distinguish between reasoning by backward chaining from queries to known facts and reasoning backwards from known consequents to unknown antecedents. To be specific, by reasoning backwards we mean that if the consequent of a rule is known to be true, we assume that the antecedent will be as well. For example, let's assume that we know that *Serves(Leaf, VegetarianFood)* is true. Since this fact matches the consequent of our rule, we might reason backwards to the conclusion that *VegetarianRestaurant(Leaf)*.

abduction

While backward chaining is a sound method of reasoning, reasoning backwards is an invalid, though frequently useful, form of *plausible reasoning*. Plausible reasoning from consequents to antecedents is known as **abduction**, and as we show in Chapter 23, is often useful in accounting for many of the inferences people make while analyzing extended discourses.

complete

While forward and backward reasoning are sound, neither is **complete**. This means that there are valid inferences that cannot be found by systems using these methods alone. Fortunately, there is an alternative inference technique called

resolution

resolution that is sound and complete. Unfortunately, inference systems based on resolution are far more computationally expensive than forward or backward chaining systems. In practice, therefore, most systems use some form of chaining and place a burden on knowledge-base developers to encode the knowledge in a fashion that permits the necessary inferences to be drawn.

16.4 Event and State Representations

Much of the semantics that we wish to capture consists of representations of states and events. States are conditions, or properties, that remain unchanged over an extended period of time, and events denote changes in some state of affairs. The representation of both states and events may involve a host of participants, props, times and locations.

The representations for events and states that we have used thus far have consisted of single predicates with as many arguments as are needed to incorporate all the roles associated with a given example. For example, the representation for *Leaf serves vegetarian fare* consists of a single predicate with arguments for the entity doing the serving and the thing served.

$$\text{Serves}(\text{Leaf}, \text{VegetarianFare}) \quad (16.38)$$

This approach assumes that the predicate used to represent an event verb has the same number of arguments as are present in the verb's syntactic subcategorization frame. Unfortunately, this is clearly not always the case. Consider the following examples of the verb *eat*:

- (16.39) I ate.
- (16.40) I ate a turkey sandwich.
- (16.41) I ate a turkey sandwich at my desk.
- (16.42) I ate at my desk.
- (16.43) I ate lunch.
- (16.44) I ate a turkey sandwich for lunch.

(16.45) I ate a turkey sandwich for lunch at my desk.

Clearly, choosing the correct number of arguments for the predicate representing the meaning of *eat* is a tricky problem. These examples introduce five distinct arguments, or roles, in an array of different syntactic forms, locations, and combinations. Unfortunately, predicates in FOL have fixed **arity** – they take a fixed number of arguments.

event variable

To address this problem, we introduce the notion of an **event variable** to allow us to make assertions about particular events. To do this, we can refactor our event predicates to have an existentially quantified variable as their first, *and only*, argument. Using this event variable, we can introduce additional predicates to represent the other information we have about the event. These predicates take an event variable as their first argument and related FOL terms as their second argument. The following formula illustrates this scheme with the meaning representation of 16.40 from our earlier discussion.

$$\exists e \text{ Eating}(e) \wedge \text{Eater}(e, \text{Speaker}) \wedge \text{Eaten}(e, \text{TurkeySandwich})$$

Here, the quantified variable *e* stands for the eating event and is used to bind the event predicate with the core information provided via the named roles *Eater* and *Eaten*. To handle the more complex examples, we simply add additional relations to capture the provided information, as in the following for 16.45.

$$\begin{aligned} \exists e \text{ Eating}(e) \wedge \text{Eater}(e, \text{Speaker}) \wedge \text{Eaten}(e, \text{TurkeySandwich}) \quad (16.46) \\ \wedge \text{Meal}(e, \text{Lunch}) \wedge \text{Location}(e, \text{Desk}) \end{aligned}$$

neo-Davidsonian

Event representations of this sort are referred to as **neo-Davidsonian** event representations (Davidson 1967, Parsons 1990) after the philosopher Donald Davidson who introduced the notion of an event variable (Davidson, 1967). To summarize, in the neo-Davidsonian approach to event representations:

- Events are captured with predicates that take a single event variable as an argument.
- There is no need to specify a fixed number of arguments for a given FOL predicate; rather, as many roles and fillers can be glued on as are provided in the input.
- No more roles are postulated than are mentioned in the input.
- The logical connections among closely related inputs that share the same predicate are satisfied without the need for additional inference.

This approach still leaves us with the problem of determining the set of predicates needed to represent roles associated with specific events like *Eater* and *Eaten*, as well as more general concepts like *Location* and *Time*. We'll return to this problem in more detail in Chapter 20.

16.4.1 Representing Time

temporal logic

In our discussion of events, we did not seriously address the issue of capturing the time when the represented events are supposed to have occurred. The representation of such information in a useful form is the domain of **temporal logic**. This discussion introduces the most basic concerns of temporal logic and briefly discusses the means by which human languages convey temporal information, which, among other things, includes **tense logic**, the ways that verb tenses convey temporal infor-

tense logic

mation. A more detailed discussion of robust approaches to the representation and analysis of temporal expressions is presented in Chapter 18.

The most straightforward theory of time holds that it flows inexorably forward and that events are associated with either points or intervals in time, as on a timeline. We can order distinct events by situating them on the timeline; one event *precedes* another if the flow of time leads from the first event to the second. Accompanying these notions in most theories is the idea of the current moment in time. Combining this notion with the idea of a temporal ordering relationship yields the familiar notions of past, present, and future.

Many schemes can represent this kind of temporal information. The one presented here is a fairly simple one that stays within the FOL framework of reified events that we have been pursuing. Consider the following examples:

- (16.47) I arrived in New York.
- (16.48) I am arriving in New York.
- (16.49) I will arrive in New York.

These sentences all refer to the same kind of event and differ solely in the tense of the verb. In our current scheme for representing events, all three would share the following kind of representation, which lacks any temporal information:

$$\exists e \text{Arriving}(e) \wedge \text{Arriver}(e, \text{Speaker}) \wedge \text{Destination}(e, \text{NewYork}) \quad (16.50)$$

The temporal information provided by the tense of the verbs can be exploited by predicing additional information about the event variable e . Specifically, we can add temporal variables representing the interval corresponding to the event, the end point of the event, and temporal predicates relating this end point to the current time as indicated by the tense of the verb. Such an approach yields the following representations for our *arriving* examples:

$$\begin{aligned} & \exists e, i, n \text{Arriving}(e) \wedge \text{Arriver}(e, \text{Speaker}) \wedge \text{Destination}(e, \text{NewYork}) \\ & \quad \wedge \text{IntervalOf}(e, i) \wedge \text{EndPoint}(i, n) \wedge \text{Precedes}(n, \text{Now}) \\ & \exists e, i, n \text{Arriving}(e) \wedge \text{Arriver}(e, \text{Speaker}) \wedge \text{Destination}(e, \text{NewYork}) \\ & \quad \wedge \text{IntervalOf}(e, i) \wedge \text{MemberOf}(i, \text{Now}) \\ & \exists e, i, n \text{Arriving}(e) \wedge \text{Arriver}(e, \text{Speaker}) \wedge \text{Destination}(e, \text{NewYork}) \\ & \quad \wedge \text{IntervalOf}(e, i) \wedge \text{EndPoint}(i, n) \wedge \text{Precedes}(\text{Now}, n) \end{aligned}$$

This representation introduces a variable to stand for the interval of time associated with the event and a variable that stands for the end of that interval. The two-place predicate *Precedes* represents the notion that the first time-point argument precedes the second in time; the constant *Now* refers to the current time. For past events, the end point of the interval must precede the current time. Similarly, for future events the current time must precede the end of the event. For events happening in the present, the current time is contained within the event interval.

Unfortunately, the relation between simple verb tenses and points in time is by no means straightforward. Consider the following examples:

- (16.51) Ok, we fly from San Francisco to Boston at 10.
- (16.52) Flight 1390 will be at the gate an hour now.

In the first example, the present tense of the verb *fly* is used to refer to a future event, while in the second the future tense is used to refer to a past event.

More complications occur when we consider some of the other verb tenses. Consider the following examples:

(16.53) Flight 1902 arrived late.

(16.54) Flight 1902 had arrived late.

reference point

Although both refer to events in the past, representing them in the same way seems wrong. The second example seems to have another unnamed event lurking in the background (e.g., Flight 1902 had already arrived late *when* something else happened). To account for this phenomena, [Reichenbach \(1947\)](#) introduced the notion of a **reference point**. In our simple temporal scheme, the current moment in time is equated with the time of the utterance and is used as a reference point for when the event occurred (before, at, or after). In Reichenbach's approach, the notion of the reference point is separated from the utterance time and the event time. The following examples illustrate the basics of this approach:

(16.55) When Mary's flight departed, I ate lunch.

(16.56) When Mary's flight departed, I had eaten lunch.

In both of these examples, the eating event has happened in the past, that is, prior to the utterance. However, the verb tense in the first example indicates that the eating event began when the flight departed, while the second example indicates that the eating was accomplished prior to the flight's departure. Therefore, in Reichenbach's terms the *departure* event specifies the reference point. These facts can be accommodated by additional constraints relating the *eating* and *departure* events. In the first example, the reference point precedes the *eating* event, and in the second example, the *eating* precedes the reference point. Figure 16.5 illustrates Reichenbach's approach with the primary English tenses. Exercise 16.6 asks you to represent these examples in FOL.

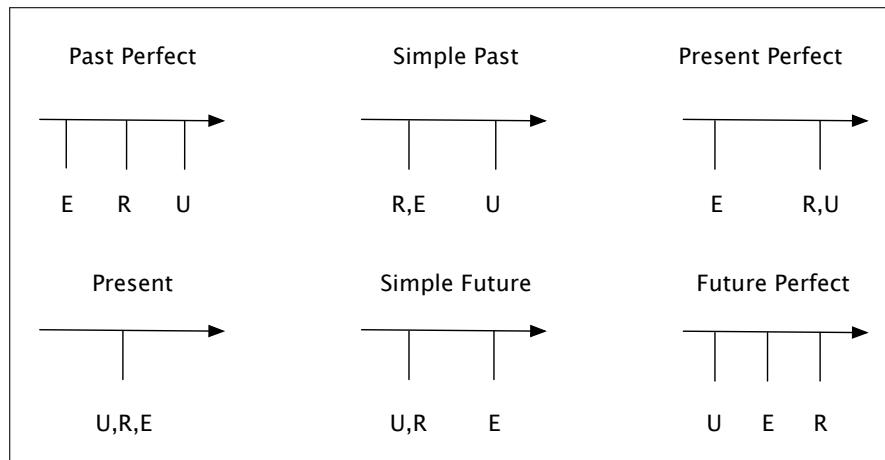


Figure 16.5 Reichenbach's approach applied to various English tenses. In these diagrams, time flows from left to right, **E** denotes the time of the event, **R** denotes the reference time, and **U** denotes the time of the utterance.

This discussion has focused narrowly on the broad notions of past, present, and future and how they are signaled by various English verb tenses. Of course, languages have many other ways to convey temporal information, including temporal expressions:

(16.57) I'd like to go at 6:45 in the morning.

(16.58) Somewhere around noon, please.

As we show in Chapter 18, grammars for such temporal expressions are of considerable practical importance to information extraction and question-answering applications.

Finally, we should note that a systematic conceptual organization is reflected in examples like these. In particular, temporal expressions in English are frequently expressed in spatial terms, as is illustrated by the various uses of *at*, *in*, *somewhere*, and *near* in these examples (Lakoff and Johnson 1980, Jackendoff 1983). Metaphorical organizations such as these, in which one domain is systematically expressed in terms of another, are very common in languages of the world.

16.4.2 Aspect

In the last section, we discussed ways to represent the time of an event with respect to the time of an utterance describing it. In this section, we address the notion of **aspect**, which concerns a cluster of related topics, including whether an event has ended or is ongoing, whether it is conceptualized as happening at a point in time or over some interval, and whether any particular state in the world comes about because of it. Based on these and related notions, event expressions have traditionally been divided into four general classes illustrated in the following examples:

Stative: I know my departure gate.

Activity: John is flying.

Accomplishment: Sally booked her flight.

Achievement: She found her gate.

Although the earliest versions of this classification were discussed by Aristotle, the one presented here is due to Vendler (1967).

Stative expressions represent the notion of an event participant having a particular property, or being in a state, at a given point in time. As such, these expressions can be thought of as capturing an aspect of a world at a single point in time. Consider the following ATIS examples.

(16.59) I like Flight 840 arriving at 10:06.

(16.60) I need the cheapest fare.

(16.61) I want to go first class.

In examples like these, the event participant denoted by the subject can be seen as experiencing something at a specific point in time. Whether or not the experiencer was in the same state earlier or will be in the future is left unspecified.

Activity expressions describe events undertaken by a participant and have no particular end point. Unlike statives, activities are seen as occurring over some span of time and are therefore not associated with single points in time. Consider the following examples:

(16.62) She drove a Mazda.

(16.63) I live in Brooklyn.

These examples both specify that the subject is engaged in, or has engaged in, the activity specified by the verb for some period of time.

The final aspectual class, **achievement expressions**, is similar to accomplishments in that these expressions result in a state. Consider the following:

(16.64) She found her gate.

(16.65) I reached New York.

Unlike accomplishments, achievement events are thought of as happening in an instant and are not equated with any particular activity leading up to the state. To be more specific, the events in these examples may have been preceded by extended *searching* or *traveling* events, but the events corresponding directly to *found* and *reach* are conceived of as points, not intervals.

Note that since both accomplishments and achievements are events that result in a state, they are sometimes characterized as subtypes of a single aspectual class. Members of this combined class are known as **telic eventualities**.

**telic
eventualities**

16.5 Description Logics

As noted at the beginning of this chapter, a fair number of representational schemes have been invented to capture the meaning of linguistic utterances. It is now widely accepted that meanings represented in these various approaches can, in principle, be translated into equivalent statements in FOL with relative ease. The difficulty is that in many of these approaches the semantics of a statement are defined procedurally. That is, the meaning arises from whatever the system that interprets it does with it.

Description logics are an effort to better specify the semantics of these earlier structured network representations and to provide a conceptual framework that is especially well suited to certain kinds of domain modeling. Formally, the term Description Logics refers to a family of logical approaches that correspond to varying subsets of FOL. The restrictions placed on the expressiveness of Description Logics serve to guarantee the tractability of various critical kinds of inference. Our focus here, however, will be on the modeling aspects of DLs rather than on computational complexity issues.

When using Description Logics to model an application domain, the emphasis is on the representation of knowledge about categories, individuals that belong to those categories, and the relationships that can hold among these individuals. The set of categories, or concepts, that make up a particular application domain is called its **terminology**. The portion of a knowledge base that contains the terminology is traditionally called the **TBox**; this is in contrast to the **ABox** that contains facts about individuals. The terminology is typically arranged into a hierarchical organization called an **ontology** that captures the subset/superset relations among the categories.

terminology
TBox
ABox
ontology

Returning to our earlier culinary domain, we represented domain concepts using unary predicates such as *Restaurant*(*x*); the DL equivalent omits the variable, so the restaurant category is simply written as **Restaurant**.² To capture the fact that a particular domain element, such as *Frasca*, is a restaurant, we assert **Restaurant(Frasca)** in much the same way we would in FOL. The semantics of these categories are specified in precisely the same way that was introduced earlier in Section 16.2: a category like **Restaurant** simply denotes the set of domain elements that are restaurants.

Once we've specified the categories of interest in a particular domain, the next step is to arrange them into a hierarchical structure. There are two ways to capture the hierarchical relationships present in a terminology: we can directly assert relations between categories that are related hierarchically, or we can provide complete definitions for our concepts and then rely on inference to provide hierarchical

² DL statements are conventionally typeset with a sans serif font. We'll follow that convention here, reverting to our standard mathematical notation when giving FOL equivalents of DL statements.

relationships. The choice between these methods hinges on the use to which the resulting categories will be put and the feasibility of formulating precise definitions for many naturally occurring categories. We'll discuss the first option here and return to the notion of definitions later in this section.

subsumption

To directly specify a hierarchical structure, we can assert **subsumption** relations between the appropriate concepts in a terminology. The subsumption relation is conventionally written as $C \sqsubseteq D$ and is read as C is subsumed by D ; that is, all members of the category C are also members of the category D . Not surprisingly, the formal semantics of this relation are provided by a simple set relation; any domain element that is in the set denoted by C is also in the set denoted by D .

Adding the following statements to the TBox asserts that all restaurants are commercial establishments and, moreover, that there are various subtypes of restaurants.

$$\text{Restaurant} \sqsubseteq \text{CommercialEstablishment} \quad (16.66)$$

$$\text{ItalianRestaurant} \sqsubseteq \text{Restaurant} \quad (16.67)$$

$$\text{ChineseRestaurant} \sqsubseteq \text{Restaurant} \quad (16.68)$$

$$\text{MexicanRestaurant} \sqsubseteq \text{Restaurant} \quad (16.69)$$

Ontologies such as this are conventionally illustrated with diagrams such as the one shown in Fig. 16.6, where subsumption relations are denoted by links between the nodes representing the categories.

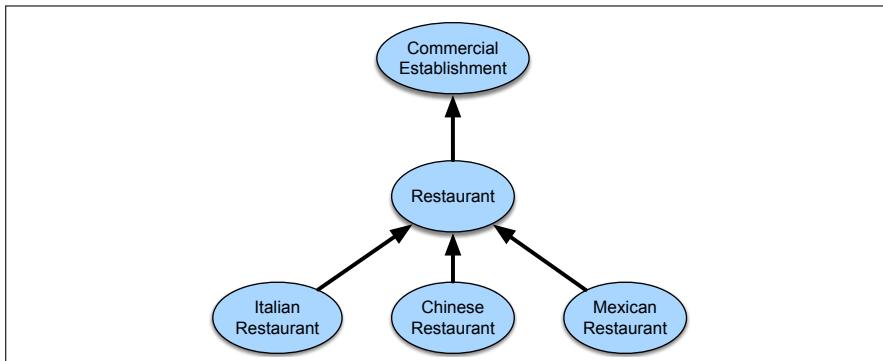


Figure 16.6 A graphical network representation of a set of subsumption relations in the restaurant domain.

Note, that it was precisely the vague nature of semantic network diagrams like this that motivated the development of Description Logics. For example, from this diagram we can't tell whether the given set of categories is exhaustive or disjoint. That is, we can't tell if these are all the kinds of restaurants that we'll be dealing with in our domain or whether there might be others. We also can't tell if an individual restaurant must fall into only *one* of these categories, or if it is possible, for example, for a restaurant to be *both* Italian and Chinese. The DL statements given above are more transparent in their meaning; they simply assert a set of subsumption relations between categories and make no claims about coverage or mutual exclusion.

If an application requires coverage and disjointness information, then such information must be made explicitly. The simplest ways to capture this kind of information is through the use of negation and disjunction operators. For example, the following assertion would tell us that Chinese restaurants can't also be Italian restaurants.

$$\text{ChineseRestaurant} \sqsubseteq \text{not ItalianRestaurant} \quad (16.70)$$

Specifying that a set of subconcepts covers a category can be achieved with disjunction, as in the following:

$$\text{Restaurant} \sqsubseteq (\text{or ItalianRestaurant ChineseRestaurant MexicanRestaurant}) \quad (16.71)$$

Having a hierarchy such as the one given in Fig. 16.6 tells us next to nothing about the concepts in it. We certainly don't know anything about what makes a restaurant a restaurant, much less Italian, Chinese, or expensive. What is needed are additional assertions about what it means to be a member of any of these categories. In Description Logics such statements come in the form of relations between the concepts being described and other concepts in the domain. In keeping with its origins in structured network representations, relations in Description Logics are typically binary and are often referred to as roles, or role-relations.

To see how such relations work, let's consider some of the facts about restaurants discussed earlier in the chapter. We'll use the `hasCuisine` relation to capture information as to what kinds of food restaurants serve and the `hasPriceRange` relation to capture how pricey particular restaurants tend to be. We can use these relations to say something more concrete about our various classes of restaurants. Let's start with our `ItalianRestaurant` concept. As a first approximation, we might say something uncontroversial like Italian restaurants serve Italian cuisine. To capture these notions, let's first add some new concepts to our terminology to represent various kinds of cuisine.

$$\begin{array}{ll} \text{MexicanCuisine} \sqsubseteq \text{Cuisine} & \text{ExpensiveRestaurant} \sqsubseteq \text{Restaurant} \\ \text{ItalianCuisine} \sqsubseteq \text{Cuisine} & \text{ModerateRestaurant} \sqsubseteq \text{Restaurant} \\ \text{ChineseCuisine} \sqsubseteq \text{Cuisine} & \text{CheapRestaurant} \sqsubseteq \text{Restaurant} \\ \text{VegetarianCuisine} \sqsubseteq \text{Cuisine} & \end{array}$$

Next, let's revise our earlier version of `ItalianRestaurant` to capture cuisine information.

$$\text{ItalianRestaurant} \sqsubseteq \text{Restaurant} \sqcap \exists \text{hasCuisine}.\text{ItalianCuisine} \quad (16.72)$$

The correct way to read this expression is that individuals in the category `ItalianRestaurant` are subsumed both by the category `Restaurant` and by an unnamed class defined by the existential clause—the set of entities that serve Italian cuisine. An equivalent statement in FOL would be

$$\begin{aligned} \forall x \text{ItalianRestaurant}(x) \rightarrow \text{Restaurant}(x) \\ \wedge (\exists y \text{Serves}(x, y) \wedge \text{ItalianCuisine}(y)) \end{aligned} \quad (16.73)$$

This FOL translation should make it clear what the DL assertions given above do and do not entail. In particular, they don't say that domain entities classified as Italian restaurants can't engage in other relations like being expensive or even serving Chinese cuisine. And critically, they don't say much about domain entities that we know do serve Italian cuisine. In fact, inspection of the FOL translation makes it clear that we cannot *infer* that any new entities belong to this category based on their characteristics. The best we can do is infer new facts about restaurants that we're explicitly told are members of this category.

Of course, inferring the category membership of individuals given certain characteristics is a common and critical reasoning task that we need to support. This brings us back to the alternative approach to creating hierarchical structures in a terminology: actually providing a definition of the categories we're creating in the form of necessary and sufficient conditions for category membership. In this case, we might explicitly provide a definition for `ItalianRestaurant` as being those restaurants that serve Italian cuisine, and `ModerateRestaurant` as being those whose price range is moderate.

$$\text{ItalianRestaurant} \equiv \text{Restaurant} \sqcap \exists \text{hasCuisine}.\text{ItalianCuisine} \quad (16.74)$$

$$\text{ModerateRestaurant} \equiv \text{Restaurant} \sqcap \text{hasPriceRange}.\text{ModeratePrices} \quad (16.75)$$

While our earlier statements provided necessary conditions for membership in these categories, these statements provide both necessary and sufficient conditions.

Finally, let's now consider the superficially similar case of vegetarian restaurants. Clearly, vegetarian restaurants are those that serve vegetarian cuisine. But they don't merely serve vegetarian fare, that's all they serve. We can accommodate this kind of constraint by adding an additional restriction in the form of a universal quantifier to our earlier description of `VegetarianRestaurants`, as follows:

$$\begin{aligned} \text{VegetarianRestaurant} &\equiv \text{Restaurant} \\ &\quad \sqcap \exists \text{hasCuisine}.\text{VegetarianCuisine} \\ &\quad \sqcap \forall \text{hasCuisine}.\text{VegetarianCuisine} \end{aligned} \quad (16.76)$$

Inference

Paralleling the focus of Description Logics on categories, relations, and individuals is a processing focus on a restricted subset of logical inference. Rather than employing the full range of reasoning permitted by FOL, DL reasoning systems emphasize the closely coupled problems of subsumption and instance checking.

subsumption

instance checking

Subsumption, as a form of inference, is the task of determining, based on the facts asserted in a terminology, whether a superset/subset relationship exists between two concepts. Correspondingly, **instance checking** asks if an individual can be a member of a particular category given the facts we know about both the individual and the terminology. The inference mechanisms underlying subsumption and instance checking go beyond simply checking for explicitly stated subsumption relations in a terminology. They must explicitly reason using the relational information asserted about the terminology to infer appropriate subsumption and membership relations.

Returning to our restaurant domain, let's add a new kind of restaurant using the following statement:

$$\text{IIFornaio} \sqsubseteq \text{ModerateRestaurant} \sqcap \exists \text{hasCuisine}.\text{ItalianCuisine} \quad (16.77)$$

Given this assertion, we might ask whether the `IIFornaio` chain of restaurants might be classified as an Italian restaurant or a vegetarian restaurant. More precisely, we can pose the following questions to our reasoning system:

$$\text{IIFornaio} \sqsubseteq \text{ItalianRestaurant} \quad (16.78)$$

$$\text{IIFornaio} \sqsubseteq \text{VegetarianRestaurant} \quad (16.79)$$

The answer to the first question is positive since `IIFornaio` meets the criteria we specified for the category `ItalianRestaurant`: it's a `Restaurant` since we explicitly



Figure 16.7 A graphical network representation of the complete set of subsumption relations in the restaurant domain given the current set of assertions in the TBox.

classified it as a `ModerateRestaurant`, which is a subtype of `Restaurant`, and it meets the `has.Cuisine` class restriction since we've asserted that directly.

The answer to the second question is negative. Recall, that our criteria for vegetarian restaurants contains two requirements: it has to serve vegetarian fare, and that's all it can serve. Our current definition for `IlFornaio` fails on both counts since we have not asserted any relations that state that `IlFornaio` serves vegetarian fare, and the relation we have asserted, `hasCuisine.ItalianCuisine`, contradicts the second criteria.

implied hierarchy

A related reasoning task, based on the basic subsumption inference, is to derive the **implied hierarchy** for a terminology given facts about the categories in the terminology. This task roughly corresponds to a repeated application of the subsumption operator to pairs of concepts in the terminology. Given our current collection of statements, the expanded hierarchy shown in Fig. 16.7 can be inferred. You should convince yourself that this diagram contains all and only the subsumption links that should be present given our current knowledge.

Instance checking is the task of determining whether a particular individual can be classified as a member of a particular category. This process takes what is known about a given individual, in the form of relations and explicit categorical statements, and then compares that information with what is known about the current terminology. It then returns a list of the *most specific* categories to which the individual can belong.

As an example of a categorization problem, consider an establishment that we're told is a restaurant and serves Italian cuisine.

```

Restaurant(Gondolier)
hasCuisine(Gondolier, ItalianCuisine)

```

Here, we're being told that the entity denoted by the term `Gondolier` is a restaurant and serves Italian food. Given this new information and the contents of our current TBox, we might reasonably like to ask if this is an Italian restaurant, if it is a vegetarian restaurant, or if it has moderate prices.

Assuming the definitional statements given earlier, we can indeed categorize the `Gondolier` as an Italian restaurant. That is, the information we've been given about it meets the necessary and sufficient conditions required for membership in this category. And as with the `IlFornaio` category, this individual fails to match the stated criteria for the `VegetarianRestaurant`. Finally, the `Gondolier` might also

turn out to be a moderately priced restaurant, but we can't tell at this point since we don't know anything about its prices. What this means is that given our current knowledge the answer to the query `ModerateRestaurant(Gondolier)` would be false since it lacks the required `hasPriceRange` relation.

The implementation of subsumption, instance checking, as well as other kinds of inferences needed for practical applications, varies according to the expressivity of the Description Logic being used. However, for a Description Logic of even modest power, the primary implementation techniques are based on satisfiability methods that in turn rely on the underlying model-based semantics introduced earlier in this chapter.

OWL and the Semantic Web

The highest-profile role for Description Logics, to date, has been as a part of the development of the Semantic Web. The Semantic Web is an ongoing effort to provide a way to formally specify the semantics of the contents of the Web (Fensel et al., 2003). A key component of this effort involves the creation and deployment of ontologies for various application areas of interest. The meaning representation language used to represent this knowledge is the **Web Ontology Language** (OWL) (McGuiness and van Harmelen, 2004). OWL embodies a Description Logic that corresponds roughly to the one we've been describing here.

Web Ontology Language

16.6 Summary

This chapter has introduced the representational approach to meaning. The following are some of the highlights of this chapter:

- A major approach to meaning in computational linguistics involves the creation of **formal meaning representations** that capture the meaning-related content of linguistic inputs. These representations are intended to bridge the gap from language to common-sense knowledge of the world.
- The frameworks that specify the syntax and semantics of these representations are called **meaning representation languages**. A wide variety of such languages are used in natural language processing and artificial intelligence.
- Such representations need to be able to support the practical computational requirements of semantic processing. Among these are the need to determine **the truth of propositions**, to support **unambiguous representations**, to represent **variables**, to support **inference**, and to be sufficiently **expressive**.
- Human languages have a wide variety of features that are used to convey meaning. Among the most important of these is the ability to convey a **predicate-argument structure**.
- **First-Order Logic** is a well-understood, computationally tractable meaning representation language that offers much of what is needed in a meaning representation language.
- Important elements of semantic representation including **states** and **events** can be captured in FOL.
- **Semantic networks** and **frames** can be captured within the FOL framework.
- Modern **Description Logics** consist of useful and computationally tractable subsets of full First-Order Logic. The most prominent use of a description

logic is the **Web Ontology Language** (OWL), used in the specification of the Semantic Web.

Bibliographical and Historical Notes

The earliest computational use of declarative meaning representations in natural language processing was in the context of question-answering systems ([Green et al. 1961](#), [Raphael 1968](#), [Lindsey 1963](#)). These systems employed ad hoc representations for the facts needed to answer questions. Questions were then translated into a form that could be matched against facts in the knowledge base. [Simmons \(1965\)](#) provides an overview of these early efforts.

[Woods \(1967\)](#) investigated the use of FOL-like representations in question answering as a replacement for the ad hoc representations in use at the time. [Woods \(1973\)](#) further developed and extended these ideas in the landmark Lunar system. Interestingly, the representations used in Lunar had both truth-conditional and procedural semantics. [Winograd \(1972\)](#) employed a similar representation based on the Micro-Planner language in his SHRDLU system.

During this same period, researchers interested in the cognitive modeling of language and memory had been working with various forms of associative network representations. [Masterman \(1957\)](#) was the first to make computational use of a semantic network-like knowledge representation, although semantic networks are generally credited to [Quillian \(1968\)](#). A considerable amount of work in the semantic network framework was carried out during this era ([Norman and Rumelhart 1975](#), [Schank 1972](#), [Wilks 1975c](#), [Wilks 1975b](#), [Kintsch 1974](#)). It was during this period that a number of researchers began to incorporate Fillmore's notion of case roles ([Fillmore, 1968](#)) into their representations. [Simmons \(1973\)](#) was the earliest adopter of case roles as part of representations for natural language processing.

Detailed analyses by [Woods \(1975\)](#) and [Brachman \(1979\)](#) aimed at figuring out what semantic networks actually mean led to the development of a number of more sophisticated network-like languages including KRL ([Bobrow and Winograd, 1977](#)) and KL-ONE ([Brachman and Schmolze, 1985](#)). As these frameworks became more sophisticated and well defined, it became clear that they were restricted variants of FOL coupled with specialized indexing inference procedures. A useful collection of papers covering much of this work can be found in [Brachman and Levesque \(1985\)](#). [Russell and Norvig \(2002\)](#) describe a modern perspective on these representational efforts.

Linguistic efforts to assign semantic structures to natural language sentences in the generative era began with the work of [Katz and Fodor \(1963\)](#). The limitations of their simple feature-based representations and the natural fit of logic to many of the linguistic problems of the day quickly led to the adoption of a variety of predicate-argument structures as preferred semantic representations ([Lakoff 1972a](#), [McCawley 1968](#)). The subsequent introduction by [Montague \(1973\)](#) of the truth-conditional model-theoretic framework into linguistic theory led to a much tighter integration between theories of formal syntax and a wide range of formal semantic frameworks. Good introductions to Montague semantics and its role in linguistic theory can be found in [Dowty et al. \(1981\)](#) and [Partee \(1976\)](#).

The representation of events as reified objects is due to [Davidson \(1967\)](#). The approach presented here, which explicitly reifies event participants, is due to [Parsons](#)

(1990).

Most current computational approaches to temporal reasoning are based on Allen's notion of temporal intervals (Allen, 1984); see Chapter 18. ter Meulen (1995) provides a modern treatment of tense and aspect. Davis (1990) describes the use of FOL to represent knowledge across a wide range of common-sense domains including quantities, space, time, and beliefs.

A recent comprehensive treatment of logic and language can be found in van Benthem and ter Meulen (1997). A classic semantics text is Lyons (1977). McCawley (1993) is an indispensable textbook covering a wide range of topics concerning logic and language. Chierchia and McConnell-Ginet (1991) also broadly covers semantic issues from a linguistic perspective. Heim and Kratzer (1998) is a more recent text written from the perspective of current generative theory.

Exercises

- 16.1** Peruse your daily newspaper for three examples of ambiguous sentences or headlines. Describe the various sources of the ambiguities.
- 16.2** Consider a domain in which the word *coffee* can refer to the following concepts in a knowledge-based system: a caffeinated or decaffeinated beverage, ground coffee used to make either kind of beverage, and the beans themselves. Give arguments as to which of the following uses of coffee are ambiguous and which are vague.
 1. I've had my coffee for today.
 2. Buy some coffee on your way home.
 3. Please grind some more coffee.
- 16.3** The following rule, which we gave as a translation for Example 16.25, is not a reasonable definition of what it means to be a vegetarian restaurant.

$$\forall x \text{VegetarianRestaurant}(x) \implies \text{Serves}(x, \text{VegetarianFood})$$

Give a FOL rule that better defines vegetarian restaurants in terms of what they serve.

- 16.4** Give FOL translations for the following sentences:
 1. Vegetarians do not eat meat.
 2. Not all vegetarians eat eggs.
- 16.5** Give a set of facts and inferences necessary to prove the following assertions:
 1. McDonald's is not a vegetarian restaurant.
 2. Some vegetarians can eat at McDonald's.

Don't just place these facts in your knowledge base. Show that they can be inferred from some more general facts about vegetarians and McDonald's.
- 16.6** For the following sentences, give FOL translations that capture the temporal relationships between the events.
 1. When Mary's flight departed, I ate lunch.
 2. When Mary's flight departed, I had eaten lunch.
- 16.7** On page 309, we gave the representation *Near(Centro, Bacaro)* as a translation for the sentence *Centro is near Bacaro*. In a truth-conditional semantics, this formula is either true or false given some model. Critique this truth-conditional approach with respect to the meaning of words like *near*.

CHAPTER

17

Computational Semantics and Semantic Parsing

Placeholder

*I am the very model of a modern Major-General,
 I've information vegetable, animal, and mineral,
 I know the kings of England, and I quote the fights historical
 From Marathon to Waterloo, in order categorical...
 Gilbert and Sullivan, Pirates of Penzance*

Imagine that you are an analyst with an investment firm that tracks airline stocks. You're given the task of determining the relationship (if any) between airline announcements of fare increases and the behavior of their stocks the next day. Historical data about stock prices is easy to come by, but what about the airline announcements? You will need to know at least the name of the airline, the nature of the proposed fare hike, the dates of the announcement, and possibly the response of other airlines. Fortunately, these can be all found in news articles like this one:

Citing high fuel prices, United Airlines said Friday it has increased fares by \$6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR Corp., immediately matched the move, spokesman Tim Wagner said. United, a unit of UAL Corp., said the increase took effect Thursday and applies to most routes where it competes against discount carriers, such as Chicago to Dallas and Denver to San Francisco.

information extraction

This chapter presents techniques for extracting limited kinds of semantic content from text. This process of **information extraction** (IE), turns the unstructured information embedded in texts into structured data, for example for populating a relational database to enable further processing.

named entity recognition

We begin with the first step in most IE tasks, finding the proper names or **named entities** in a text. The task of **named entity recognition** (NER) is to find each **mention** of a named entity in the text and label its type. What constitutes a named entity type is task specific; people, places, and organizations are common, but gene or protein names (Cohen and Demner-Fushman, 2014) or financial asset classes might be relevant for some tasks. Once all the named entities in a text have been extracted, they can be linked together in sets corresponding to real-world entities, inferring, for example, that mentions of *United Airlines* and *United* refer to the same company. This is the joint task of **coreference resolution** and **entity linking** which we defer til Chapter 22.

relation extraction

Next, we turn to the task of **relation extraction**: finding and classifying semantic relations among the text entities. These are often binary relations like child-of, employment, part-whole, and geospatial relations. Relation extraction has close links to populating a relational database.

event extraction

Finally, we discuss three tasks related to *events*. **Event extraction** is finding events in which these entities participate, like, in our sample text, the fare increases

by *United* and *American* and the reporting events *said* and *cite*. **Event coreference** (Chapter 22) is needed to figure out which event mentions in a text refer to the same event; in our running example the two instances of *increase* and the phrase *the move* all refer to the same event.

temporal expression To figure out *when* the events in a text happened we extract **temporal expressions** like days of the week (*Friday* and *Thursday*), relative expressions like *two days from now* or *next year* and times such as *3:30 P.M.*. These expressions must be **normalized** onto specific calendar dates or times of day to situate events in time. In our sample task, this will allow us to link *Friday* to the time of United's announcement, and *Thursday* to the previous day's fare increase, and produce a timeline in which United's announcement follows the fare increase and American's announcement follows both of those events.

template filling Finally, many texts describe recurring stereotypical events or situations. The task of **template filling** is to find such situations in documents and fill in the template slots. These slot-fillers may consist of text segments extracted directly from the text, or concepts like times, amounts, or ontology entities that have been inferred from text elements through additional processing.

Our airline text is an example of this kind of stereotypical situation since airlines often raise fares and then wait to see if competitors follow along. In this situation, we can identify *United* as a lead airline that initially raised its fares, \$6 as the amount, *Thursday* as the increase date, and *American* as an airline that followed along, leading to a filled template like the following.

FARE-RAISE ATTEMPT:	[LEAD AIRLINE: UNITED AIRLINES AMOUNT: \$6 EFFECTIVE DATE: 2006-10-26 FOLLOWER: AMERICAN AIRLINES]
---------------------	---

18.1 Named Entity Recognition

named entity The first step in information extraction is to detect the **entities** in the text. A **named entity** is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization. The term is commonly extended to include things that aren't entities per se, including dates, times, and other kinds of **temporal expressions**, and even **numerical expressions** like prices. Here's the sample text introduced earlier with the named entities marked:

Citing high fuel prices, [ORG **United Airlines**] said [TIME **Friday**] it has increased fares by [MONEY **\$6**] per round trip on flights to some cities also served by lower-cost carriers. [ORG **American Airlines**], a unit of [ORG **AMR Corp.**], immediately matched the move, spokesman [PER **Tim Wagner**] said. [ORG **United**], a unit of [ORG **UAL Corp.**], said the increase took effect [TIME **Thursday**] and applies to most routes where it competes against discount carriers, such as [LOC **Chicago**] to [LOC **Dallas**] and [LOC **Denver**] to [LOC **San Francisco**].

The text contains 13 mentions of named entities including 5 organizations, 4 locations, 2 times, 1 person, and 1 mention of money.

In addition to their use in extracting events and the relationship between participants, named entities are useful for many other language processing tasks. In

sentiment analysis we might want to know a consumer’s sentiment toward a particular entity. Entities are a useful first stage in question answering, or for linking text to information in structured knowledge sources like Wikipedia.

Figure 18.1 shows typical generic named entity types. Many applications will also need to use specific entity types like proteins, genes, commercial products, or works of art.

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	Turing is a giant of computer science.
Organization	ORG	companies, sports teams	The IPCC warned about the cyclone.
Location	LOC	regions, mountains, seas	The Mt. Sanitas loop is in Sunshine Canyon.
Geo-Political Entity	GPE	countries, states, provinces	Palo Alto is raising the fees for parking.
Facility	FAC	bridges, buildings, airports	Consider the Golden Gate Bridge.
Vehicles	VEH	planes, trains, automobiles	It was a classic Ford Falcon.

Figure 18.1 A list of generic named entity types with the kinds of entities they refer to.

Named entity recognition means finding spans of text that constitute proper names and then classifying the type of the entity. Recognition is difficult partly because of the ambiguity of segmentation; we need to decide what’s an entity and what isn’t, and where the boundaries are. Another difficulty is caused by type ambiguity. The mention JFK can refer to a person, the airport in New York, or any number of schools, bridges, and streets around the United States. Some examples of this kind of cross-type confusion are given in Figures 18.2 and 18.3.

Name	Possible Categories
Washington	Person, Location, Political Entity, Organization, Vehicle
Downing St.	Location, Organization
IRA	Person, Organization, Monetary Instrument
Louis Vuitton	Person, Organization, Commercial Product

Figure 18.2 Common categorical ambiguities associated with various proper names.

[PER Washington] was born into slavery on the farm of James Burroughs.
 [ORG Washington] went up 2 games to 1 in the four-game series.
 Blair arrived in [LOC Washington] for what may well be his last state visit.
 In June, [GPE Washington] passed a primary seatbelt law.
 The [VEH Washington] had proved to be a leaky ship, every passage I made...

Figure 18.3 Examples of type ambiguities in the use of the name Washington.

18.1.1 NER as Sequence Labeling

The standard algorithm for named entity recognition is as a word-by-word sequence labeling task, in which the assigned tags capture both the boundary and the type. A sequence classifier like an MEMM/CRF, a bi-LSTM, or a transformer is trained to label the tokens in a text with tags that indicate the presence of particular kinds of named entities. Consider the following simplified excerpt from our running example.

[ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said.

IOB Figure 18.4 shows the same excerpt represented with **IOB** tagging. In IOB tagging we introduce a tag for the beginning (B) and inside (I) of each entity type, and one for tokens outside (O) any entity. The number of tags is thus $2n + 1$ tags, where n is the number of entity types. IOB tagging can represent exactly the same information as the bracketed notation.

Words	IOB Label	IO Label
American	B-ORG	I-ORG
Airlines	I-ORG	I-ORG
,	O	O
a	O	O
unit	O	O
of	O	O
AMR	B-ORG	I-ORG
Corp.	I-ORG	I-ORG
,	O	O
immediately	O	O
matched	O	O
the	O	O
move	O	O
,	O	O
spokesman	O	O
Tim	B-PER	I-PER
Wagner	I-PER	I-PER
said	O	O
.	O	O

Figure 18.4 Named entity tagging as a sequence model, showing IOB and IO encodings.

We've also shown IO tagging, which loses some information by eliminating the B tag. Without the B tag IO tagging is unable to distinguish between two entities of the same type that are right next to each other. Since this situation doesn't arise very often (usually there is at least some punctuation or other delimiter), IO tagging may be sufficient, and has the advantage of using only $n + 1$ tags.

In the following three sections we introduce the three standard families of algorithms for NER tagging: feature based (MEMM/CRF), neural (bi-LSTM), and rule-based.

18.1.2 A feature-based algorithm for NER

identity of w_i , identity of neighboring words
embeddings for w_i , embeddings for neighboring words
part of speech of w_i , part of speech of neighboring words
base-phrase syntactic chunk label of w_i and neighboring words
presence of w_i in a gazetteer
w_i contains a particular prefix (from all prefixes of length ≤ 4)
w_i contains a particular suffix (from all suffixes of length ≤ 4)
w_i is all upper case
word shape of w_i , word shape of neighboring words
short word shape of w_i , short word shape of neighboring words
presence of hyphen

Figure 18.5 Typical features for a feature-based NER system.

The first approach is to extract features and train an MEMM or CRF sequence model of the type we saw for part-of-speech tagging in Chapter 8. Figure 18.5 lists standard features used in such feature-based systems. We've seen many of these features before in the context of part-of-speech tagging, particularly for tagging unknown words. This is not surprising, as many unknown words are in fact named entities. Word shape features are thus particularly important in the context of NER.

word shape Recall that **word shape** features are used to represent the abstract letter pattern of the word by mapping lower-case letters to 'x', upper-case to 'X', numbers to 'd', and retaining punctuation. Thus for example I.M.F would map to X.X.X. and DC10-30 would map to XXdd-dd. A second class of shorter word shape features is also used. In these features consecutive character types are removed, so DC10-30 would be mapped to Xd-d but I.M.F would still map to X.X.X. This feature by itself accounts for a considerable part of the success of feature-based NER systems for English news text. Shape features are also particularly important in recognizing names of proteins and genes in biological texts.

For example the named entity token *L'Occitane* would generate the following non-zero valued feature values:

$\text{prefix}(w_i) = \text{L}$	$\text{suffix}(w_i) = \text{tane}$
$\text{prefix}(w_i) = \text{L}'$	$\text{suffix}(w_i) = \text{ane}$
$\text{prefix}(w_i) = \text{L}'\text{O}$	$\text{suffix}(w_i) = \text{ne}$
$\text{prefix}(w_i) = \text{L}'\text{Oc}$	$\text{suffix}(w_i) = \text{e}$
$\text{word-shape}(w_i) = \text{X}'\text{XXXXXXXX}$	$\text{short-word-shape}(w_i) = \text{X}'\text{Xx}$

gazetteer

A **gazetteer** is a list of place names, often providing millions of entries for locations with detailed geographical and political information.¹ A related resource is **name-lists**; the United States Census Bureau also provides extensive lists of first names and surnames derived from its decadal census in the U.S.² Similar lists of corporations, commercial products, and all manner of things biological and mineral are also available from a variety of sources. Gazetteer and name features are typically implemented as a binary feature for each name list. Unfortunately, such lists can be difficult to create and maintain, and their usefulness varies considerably. While gazetteers can be quite effective, lists of persons and organizations are not always helpful (Mikheev et al., 1999).

Feature effectiveness depends on the application, genre, media, and language. For example, shape features, critical for English newswire texts, are of little use with automatic speech recognition transcripts, or other non-edited or informally-edited sources, or for languages like Chinese that don't use orthographic case. The features in Fig. 18.5 should therefore be thought of as only a starting point.

Figure 18.6 illustrates the result of adding part-of-speech tags, syntactic base-phrase chunk tags, and some shape information to our earlier example.

Given such a training set, a sequence classifier like an MEMM can be trained to label new sentences. Figure 18.7 illustrates the operation of such a sequence labeler at the point where the token *Corp.* is next to be labeled. If we assume a context window that includes the two preceding and following words, then the features available to the classifier are those shown in the boxed area.

¹ www.geonames.org

² www.census.gov

Word	POS	Chunk	Short shape	Label
American	NNP	B-NP	Xx	B-ORG
Airlines	NNPS	I-NP	Xx	I-ORG
,	,	O	,	O
a	DT	B-NP	x	O
unit	NN	I-NP	x	O
of	IN	B-PP	x	O
AMR	NNP	B-NP	X	B-ORG
Corp.	NNP	I-NP	Xx.	I-ORG
,	,	O	,	O
immediately	RB	B-ADVP	x	O
matched	VBD	B-VP	x	O
the	DT	B-NP	x	O
move	NN	I-NP	x	O
,	,	O	,	O
spokesman	NN	B-NP	x	O
Tim	NNP	I-NP	Xx	B-PER
Wagner	NNP	I-NP	Xx	I-PER
said	VBD	B-VP	x	O
.	,	O	.	O

Figure 18.6 Word-by-word feature encoding for NER.**Figure 18.7** Named entity recognition as sequence labeling. The features available to the classifier during training and classification are those in the boxed area.

18.1.3 A neural algorithm for NER

The standard neural algorithm for NER is based on the bi-LSTM introduced in Chapter 9. Recall that in that model, word and character embeddings are computed for input word w_i . These are passed through a left-to-right LSTM and a right-to-left LSTM, whose outputs are concatenated (or otherwise combined) to produce a single output layer at position i . In the simplest method, this layer can then be directly passed onto a softmax that creates a probability distribution over all NER tags, and the most likely tag is chosen as t_i .

For named entity tagging this greedy approach to decoding is insufficient, since it doesn't allow us to impose the strong constraints neighboring tokens have on each other (e.g., the tag I-PER must follow another I-PER or B-PER). Instead a CRF layer is normally used on top of the bi-LSTM output, and the Viterbi decoding algorithm is used to decode. Fig. 18.8 shows a sketch of the algorithm

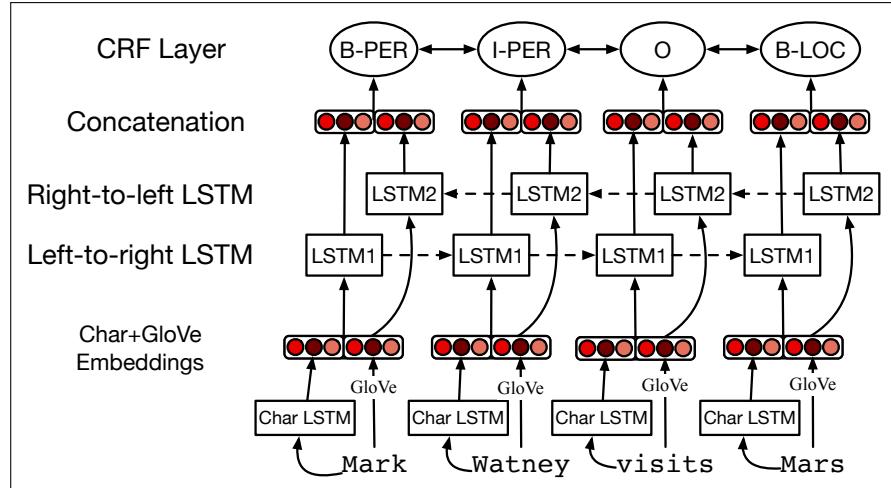


Figure 18.8 Putting it all together: character embeddings and words together in a bi-LSTM sequence model. After [Lample et al. \(2016b\)](#).

18.1.4 Rule-based NER

While machine learned (neural or MEMM/CRF) sequence models are the norm in academic research, commercial approaches to NER are often based on pragmatic combinations of lists and rules, with some smaller amount of supervised machine learning ([Chiticariu et al., 2013](#)). For example IBM System T is a text understanding architecture in which a user specifies complex declarative constraints for tagging tasks in a formal query language that includes regular expressions, dictionaries, semantic constraints, NLP operators, and table structures, all of which the system compiles into an efficient extractor ([Chiticariu et al., 2018](#)).

One common approach is to make repeated rule-based passes over a text, allowing the results of one pass to influence the next. The stages typically first involve the use of rules that have extremely high precision but low recall. Subsequent stages employ more error-prone statistical methods that take the output of the first pass into account.

1. First, use high-precision rules to tag unambiguous entity mentions.
2. Then, search for substring matches of the previously detected names.
3. Consult application-specific name lists to identify likely name entity mentions from the given domain.
4. Finally, apply probabilistic sequence labeling techniques that make use of the tags from previous stages as additional features.

The intuition behind this staged approach is twofold. First, some of the entity mentions in a text will be more clearly indicative of a given entity's class than others. Second, once an unambiguous entity mention is introduced into a text, it is likely that subsequent shortened versions will refer to the same entity (and thus the same type of entity).

18.1.5 Evaluation of Named Entity Recognition

The familiar metrics of **recall**, **precision**, and **F_1 measure** are used to evaluate NER systems. Remember that recall is the ratio of the number of correctly labeled responses to the total that should have been labeled; precision is the ratio of the num-

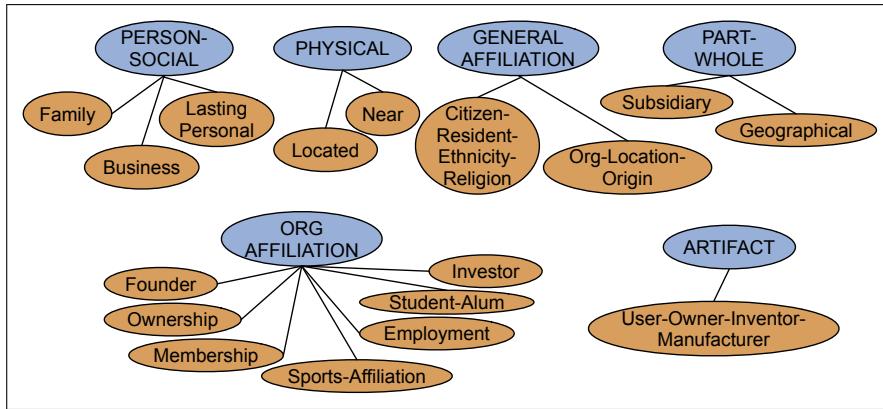


Figure 18.9 The 17 relations used in the ACE relation extraction task.

ber of correctly labeled responses to the total labeled; and *F*-measure is the harmonic mean of the two. For named entities, the *entity* rather than the word is the unit of response. Thus in the example in Fig. 18.6, the two entities *Tim Wagner* and *AMR Corp.* and the non-entity *said* would each count as a single response.

The fact that named entity tagging has a segmentation component which is not present in tasks like text categorization or part-of-speech tagging causes some problems with evaluation. For example, a system that labeled *American* but not *American Airlines* as an organization would cause two errors, a false positive for O and a false negative for I-ORG. In addition, using entities as the unit of response but words as the unit of training means that there is a mismatch between the training and test conditions.

18.2 Relation Extraction

Next on our list of tasks is to discern the relationships that exist among the detected entities. Let's return to our sample airline text:

Citing high fuel prices, [ORG United Airlines] said [TIME Friday] it has increased fares by [MONEY \$6] per round trip on flights to some cities also served by lower-cost carriers. [ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said. [ORG United], a unit of [ORG UAL Corp.], said the increase took effect [TIME Thursday] and applies to most routes where it competes against discount carriers, such as [LOC Chicago] to [LOC Dallas] and [LOC Denver] to [LOC San Francisco].

The text tells us, for example, that *Tim Wagner* is a spokesman for *American Airlines*, that *United* is a unit of *UAL Corp.*, and that *American* is a unit of *AMR*. These binary relations are instances of more generic relations such as **part-of** or **employs** that are fairly frequent in news-style texts. Figure 18.9 lists the 17 relations used in the ACE relation extraction evaluations and Fig. 18.10 shows some sample relations. We might also extract more domain-specific relation such as the notion of an airline route. For example from this text we can conclude that *United* has routes to *Chicago*, *Dallas*, *Denver*, and *San Francisco*.

Relations	Types	Examples
Physical-Located	PER-GPE	He was in Tennessee
Part-Whole-Subsidiary	ORG-ORG	XYZ, the parent company of ABC
Person-Social-Family	PER-PER	Yoko's husband John
Org-AFF-Founder	PER-ORG	Steve Jobs, co-founder of Apple...

Figure 18.10 Semantic relations with examples and the named entity types they involve.

Domain		$\mathcal{D} = \{a, b, c, d, e, f, g, h, i\}$
United, UAL, American Airlines, AMR		a, b, c, d
Tim Wagner		e
Chicago, Dallas, Denver, and San Francisco		f, g, h, i
Classes		
United, UAL, American, and AMR are organizations		$Org = \{a, b, c, d\}$
Tim Wagner is a person		$Pers = \{e\}$
Chicago, Dallas, Denver, and San Francisco are places		$Loc = \{f, g, h, i\}$
Relations		
United is a unit of UAL		$PartOf = \{\langle a, b \rangle, \langle c, d \rangle\}$
American is a unit of AMR		
Tim Wagner works for American Airlines		$OrgAff = \{\langle c, e \rangle\}$
United serves Chicago, Dallas, Denver, and San Francisco		$Serves = \{\langle a, f \rangle, \langle a, g \rangle, \langle a, h \rangle, \langle a, i \rangle\}$

Figure 18.11 A model-based view of the relations and entities in our sample text.

These relations correspond nicely to the model-theoretic notions we introduced in Chapter 16 to ground the meanings of the logical forms. That is, a relation consists of a set of ordered tuples over elements of a domain. In most standard information-extraction applications, the domain elements correspond to the named entities that occur in the text, to the underlying entities that result from co-reference resolution, or to entities selected from a domain ontology. Figure 18.11 shows a model-based view of the set of entities and relations that can be extracted from our running example. Notice how this model-theoretic view subsumes the NER task as well; named entity recognition corresponds to the identification of a class of unary relations.

Sets of relations have been defined for many other domains as well. For example UMLS, the Unified Medical Language System from the US National Library of Medicine has a network that defines 134 broad subject categories, entity types, and 54 relations between the entities, such as the following:

Entity	Relation	Entity
Injury	disrupts	Physiological Function
Bodily Location	location-of	Biologic Function
Anatomical Structure	part-of	Organism
Pharmacologic Substance	causes	Pathological Function
Pharmacologic Substance	treats	Pathologic Function

Given a medical sentence like this one:

- (18.1) Doppler echocardiography can be used to diagnose left anterior descending artery stenosis in patients with type 2 diabetes

We could thus extract the UMLS relation:

Echocardiography, Doppler Diagnoses Acquired stenosis

infoboxes

Wikipedia also offers a large supply of relations, drawn from **infoboxes**, structured tables associated with certain Wikipedia articles. For example, the Wikipedia

RDF infobox for **Stanford** includes structured facts like `state = "California"` or `president = "Mark Tessier-Lavigne"`. These facts can be turned into relations like `president-of` or `located-in`, or into relations in a metalanguage called **RDF** (Resource Description Framework). An **RDF triple** is a tuple of entity-relation-entity, called a subject-predicate-object expression. Here's a sample RDF triple:

subject	predicate	object
Golden Gate Park	location	San Francisco

Freebase For example the crowdsourced DBpedia (Bizer et al., 2009) is an ontology derived from Wikipedia containing over 2 billion RDF triples. Another dataset from Wikipedia infoboxes, **Freebase** (Bollacker et al., 2008), now part of Wikidata (Vrandečić and Krötzsch, 2014), has relations like:

people/person/nationality		
location/location/contains		

is-a WordNet or other ontologies offer useful ontological relations that express hierarchical relations between words or concepts. For example WordNet has the **is-a** or **hypernym** relation between classes,

Giraffe is-a ruminant is-a ungulate is-a mammal is-a vertebrate ...

WordNet also has *Instance-of* relation between individuals and classes, so that for example *San Francisco* is in the *Instance-of* relation with *city*. Extracting these relations is an important step in extending or building ontologies.

There are five main classes of algorithms for relation extraction: **handwritten patterns**, **supervised machine learning**, **semi-supervised** (via **bootstrapping** and via **distant supervision**), and **unsupervised**. We'll introduce each of these in the next sections.

18.2.1 Using Patterns to Extract Relations

The earliest and still common algorithm for relation extraction is lexico-syntactic patterns, first developed by Hearst (1992a). Consider the following sentence:

Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.

Hearst points out that most human readers will not know what *Gelidium* is, but that they can readily infer that it is a kind of (a **hyponym** of) *red algae*, whatever that is. She suggests that the following **lexico-syntactic pattern**

$$NP_0 \text{ such as } NP_1 \{, NP_2 \dots, (\text{and}|\text{or})NP_i\}, i \geq 1 \quad (18.2)$$

implies the following semantics

$$\forall NP_i, i \geq 1, \text{hyponym}(NP_i, NP_0) \quad (18.3)$$

allowing us to infer

$$\text{hyponym}(\text{Gelidium}, \text{red algae}) \quad (18.4)$$

Figure 18.12 shows five patterns Hearst (1992a, 1998) suggested for inferring the hyponym relation; we've shown NP_H as the parent/hyponym. Modern versions of the pattern-based approach extend it by adding named entity constraints. For example if our goal is to answer questions about “Who holds what office in which organization?”, we can use patterns like the following:

$NP \{, NP\}^* \{, \}$ (and or) other NP_H	temples, treasures, and other important civic buildings
NP_H such as $\{NP_j\}^* \{(or and)\} NP$	red algae such as Gelidium
such NP_H as $\{NP_j\}^* \{(or and)\} NP$	such authors as Herrick, Goldsmith, and Shakespeare
$NP_H \{, \}$ including $\{NP_j\}^* \{(or and)\} NP$	common-law countries , including Canada and England
$NP_H \{, \}$ especially $\{NP_j\}^* \{(or and)\} NP$	European countries , especially France, England, and Spain

Figure 18.12 Hand-built lexico-syntactic patterns for finding hypernyms, using {} to mark optionality (Hearst 1992a, Hearst 1998).

PER, POSITION of ORG:

George Marshall, Secretary of State of the United States

PER (named|appointed|chose|etc.) **PER** **Prep?** **POSITION**

Truman appointed Marshall Secretary of State

PER [be]? (named|appointed|etc.) **Prep?** **ORG POSITION**

George Marshall was named US Secretary of State

Hand-built patterns have the advantage of high-precision and they can be tailored to specific domains. On the other hand, they are often low-recall, and it's a lot of work to create them for all possible patterns.

18.2.2 Relation Extraction via Supervised Learning

Supervised machine learning approaches to relation extraction follow a scheme that should be familiar by now. A fixed set of relations and entities is chosen, a training corpus is hand-annotated with the relations and entities, and the annotated texts are then used to train classifiers to annotate an unseen test set.

The most straightforward approach has three steps, illustrated in Fig. 18.13. Step one is to find pairs of named entities (usually in the same sentence). In step two, a filtering classifier is trained to make a binary decision as to whether a given pair of named entities are related (by any relation). Positive examples are extracted directly from all relations in the annotated corpus, and negative examples are generated from within-sentence entity pairs that are not annotated with a relation. In step 3, a classifier is trained to assign a label to the relations that were found by step 2. The use of the filtering classifier can speed up the final classification and also allows the use of distinct feature-sets appropriate for each task. For each of the two classifiers, we can use any of the standard classification techniques (logistic regression, neural network, SVM, etc.)

```

function FINDRELATIONS(words) returns relations
    relations  $\leftarrow$  nil
    entities  $\leftarrow$  FINDENTITIES(words)
    forall entity pairs  $\langle e_1, e_2 \rangle$  in entities do
        if RELATED?(e1, e2)
            relations  $\leftarrow$  relations+CLASSIFYRELATION(e1, e2)
    
```

Figure 18.13 Finding and classifying the relations among entities in a text.

For the feature-based classifiers like logistic regression or random forests the most important step is to identify useful features. Let's consider features for clas-

sifying the relationship between *American Airlines* (Mention 1, or M1) and *Tim Wagner* (Mention 2, M2) from this sentence:

(18.5) **American Airlines**, a unit of AMR, immediately matched the move,
spokesman **Tim Wagner** said

Useful word features include

- The headwords of M1 and M2 and their concatenation
Airlines Wagner Airlines-Wagner
- Bag-of-words and bigrams in M1 and M2
American, Airlines, Tim, Wagner, American Airlines, Tim Wagner
- Words or bigrams in particular positions
M2: **-1 spokesman**
M2: **+1 said**
- Bag of words or bigrams between M1 and M2:
a, AMR, of, immediately, matched, move, spokesman, the, unit
- Stemmed versions of the same

Embeddings can be used to represent words in any of these features. Useful named entity features include

- Named-entity types and their concatenation
(M1: **ORG**, M2: **PER**, M1M2: **ORG-PER**)
- Entity Level of M1 and M2 (from the set NAME, NOMINAL, PRONOUN)
M1: **NAME** [it or he would be **PRONOUN**]
M2: **NAME** [**the company** would be **NOMINAL**]
- Number of entities between the arguments (in this case **1**, for AMR)

The **syntactic structure** of a sentence can also signal relationships among its entities. Syntax is often featured by using strings representing **syntactic paths**: the (dependency or constituency) path traversed through the tree in getting from one entity to the other.

- Base syntactic chunk sequence from M1 to M2
NP NP PP VP NP NP
- Constituent paths between M1 and M2
NP ↑ NP ↑ S ↑ S ↓ NP
- Dependency-tree paths
Airlines ←_{subj} matched ←_{comp} said →_{subj} Wagner

Figure 18.14 summarizes many of the features we have discussed that could be used for classifying the relationship between *American Airlines* and *Tim Wagner* from our example text.

Neural models for relation extraction similarly treat the task as supervised classification. One option is to use a similar architecture as we saw for named entity tagging: a bi-LSTM model with word embeddings as inputs and a single softmax classification of the sentence output as a 1-of-N relation label. Because relations often hold between entities that are far part in a sentence (or across sentences), it may be possible to get higher performance from algorithms like convolutional nets (dos Santos et al., 2015) or chain or tree LSTMS (Miwa and Bansal 2016, Peng et al. 2017).

In general, if the test set is similar enough to the training set, and if there is enough hand-labeled data, supervised relation extraction systems can get high accuracies. But labeling a large training set is extremely expensive and supervised

M1 headword	<i>airlines</i> (as a word token or an embedding)
M2 headword	<i>Wagner</i>
Word(s) before M1	NONE
Word(s) after M2	<i>said</i>
Bag of words between	{ <i>a, unit, of, AMR, Inc., immediately, matched, the, move, spokesman</i> }
M1 type	ORG
M2 type	PERS
Concatenated types	ORG-PERS
Constituent path	$NP \uparrow NP \uparrow S \uparrow S \downarrow NP$
Base phrase path	$NP \rightarrow NP \rightarrow PP \rightarrow NP \rightarrow VP \rightarrow NP \rightarrow NP$
Typed-dependency path	<i>Airlines</i> \leftarrow_{subj} <i>matched</i> \leftarrow_{comp} <i>said</i> \rightarrow_{subj} <i>Wagner</i>

Figure 18.14 Sample of features extracted during classification of the <American Airlines, Tim Wagner> tuple; M1 is the first mention, M2 the second.

models are brittle: they don't generalize well to different text genres. For this reason, much research in relation extraction has focused on the semi-supervised and unsupervised approaches we turn to next.

18.2.3 Semisupervised Relation Extraction via Bootstrapping

Supervised machine learning assumes that we have lots of labeled data. Unfortunately, this is expensive. But suppose we just have a few high-precision **seed patterns**, like those in Section 18.2.1, or perhaps a few **seed tuples**. That's enough to bootstrap a classifier! **Bootstrapping** proceeds by taking the entities in the seed pair, and then finding sentences (on the web, or whatever dataset we are using) that contain both entities. From all such sentences, we extract and generalize the context around the entities to learn new patterns. Fig. 18.15 sketches a basic algorithm.

```

function BOOTSTRAP(Relation R) returns new relation tuples
  tuples  $\leftarrow$  Gather a set of seed tuples that have relation R
  iterate
    sentences  $\leftarrow$  find sentences that contain entities in tuples
    patterns  $\leftarrow$  generalize the context between and around entities in sentences
    newpairs  $\leftarrow$  use patterns to grep for more tuples
    newpairs  $\leftarrow$  newpairs with high confidence
    tuples  $\leftarrow$  tuples + newpairs
  return tuples

```

Figure 18.15 Bootstrapping from seed entity pairs to learn relations.

Suppose, for example, that we need to create a list of airline/hub pairs, and we know only that Ryanair has a hub at Charleroi. We can use this seed fact to discover new patterns by finding other mentions of this relation in our corpus. We search for the terms *Ryanair*, *Charleroi* and *hub* in some proximity. Perhaps we find the following set of sentences:

- (18.6) Budget airline Ryanair, which uses Charleroi as a hub, scrapped all weekend flights out of the airport.
- (18.7) All flights in and out of Ryanair's Belgian hub at Charleroi airport were grounded on Friday...

- (18.8) A spokesman at Charleroi, a main hub for Ryanair, estimated that 8000 passengers had already been affected.

From these results, we can use the context of words between the entity mentions, the words before mention one, the word after mention two, and the named entity types of the two mentions, and perhaps other features, to extract general patterns such as the following:

```
/ [ORG], which uses [LOC] as a hub /
/ [ORG]'s hub at [LOC] /
/ [LOC] a main hub for [ORG] /
```

These new patterns can then be used to search for additional tuples.

**confidence values
semantic drift**

Bootstrapping systems also assign **confidence values** to new tuples to avoid **semantic drift**. In semantic drift, an erroneous pattern leads to the introduction of erroneous tuples, which, in turn, lead to the creation of problematic patterns and the meaning of the extracted relations ‘drifts’. Consider the following example:

- (18.9) Sydney has a ferry hub at Circular Quay.

If accepted as a positive example, this expression could lead to the incorrect introduction of the tuple $\langle \text{Sydney}, \text{CircularQuay} \rangle$. Patterns based on this tuple could propagate further errors into the database.

Confidence values for patterns are based on balancing two factors: the pattern’s performance with respect to the current set of tuples and the pattern’s productivity in terms of the number of matches it produces in the document collection. More formally, given a document collection \mathcal{D} , a current set of tuples T , and a proposed pattern p , we need to track two factors:

- *hits*: the set of tuples in T that p matches while looking in \mathcal{D}
- *finds*: The total set of tuples that p finds in \mathcal{D}

The following equation balances these considerations (Riloff and Jones, 1999).

$$\text{Conf}_{R\log F}(p) = \frac{\text{hits}_p}{\text{finds}_p} \times \log(\text{finds}_p) \quad (18.10)$$

This metric is generally normalized to produce a probability.

noisy-or

We can assess the confidence in a proposed new tuple by combining the evidence supporting it from all the patterns P' that match that tuple in \mathcal{D} (Agichtein and Gravano, 2000). One way to combine such evidence is the **noisy-or** technique. Assume that a given tuple is supported by a subset of the patterns in P , each with its own confidence assessed as above. In the noisy-or model, we make two basic assumptions. First, that for a proposed tuple to be false, *all* of its supporting patterns must have been in error, and second, that the sources of their individual failures are all independent. If we loosely treat our confidence measures as probabilities, then the probability of any individual pattern p failing is $1 - \text{Conf}(p)$; the probability of all of the supporting patterns for a tuple being wrong is the product of their individual failure probabilities, leaving us with the following equation for our confidence in a new tuple.

$$\text{Conf}(t) = 1 - \prod_{p \in P'} (1 - \text{Conf}(p)) \quad (18.11)$$

Setting conservative confidence thresholds for the acceptance of new patterns and tuples during the bootstrapping process helps prevent the system from drifting away from the targeted relation.

18.2.4 Distant Supervision for Relation Extraction

distant supervision

Although text that has been hand-labeled with relation labels is extremely expensive to produce, there are ways to find indirect sources of training data. The **distant supervision** method of Mintz et al. (2009) combines the advantages of bootstrapping with supervised learning. Instead of just a handful of seeds, distant supervision uses a large database to acquire a huge number of seed examples, creates lots of noisy pattern features from all these examples and then combines them in a supervised classifier.

For example suppose we are trying to learn the *place-of-birth* relationship between people and their birth cities. In the seed-based approach, we might have only 5 examples to start with. But Wikipedia-based databases like DBpedia or Freebase have tens of thousands of examples of many relations; including over 100,000 examples of *place-of-birth*, (<Edwin Hubble, Marshfield>, <Albert Einstein, Ulm>, etc.). The next step is to run named entity taggers on large amounts of text—Mintz et al. (2009) used 800,000 articles from Wikipedia—and extract all sentences that have two named entities that match the tuple, like the following:

...Hubble was born in Marshfield...
 ...Einstein, born (1879), Ulm...
 ...Hubble's birthplace in Marshfield...

Training instances can now be extracted from this data, one training instance for each identical tuple `<relation, entity1, entity2>`. Thus there will be one training instance for each of:

`<born-in, Edwin Hubble, Marshfield>`
`<born-in, Albert Einstein, Ulm>`
`<born-year, Albert Einstein, 1879>`

and so on.

We can then apply feature-based or neural classification. For feature-based classification, standard supervised relation extraction features like the named entity labels of the two mentions, the words and dependency paths in between the mentions, and neighboring words. Each tuple will have features collected from many training instances; the feature vector for a single training instance like (`<born-in, Albert Einstein, Ulm>`) will have lexical and syntactic features from many different sentences that mention Einstein and Ulm.

Because distant supervision has very large training sets, it is also able to use very rich features that are conjunctions of these individual features. So we will extract thousands of patterns that conjoin the entity types with the intervening words or dependency paths like these:

PER was born in LOC
 PER, born (XXXX), LOC
 PER's birthplace in LOC

To return to our running example, for this sentence:

(18.12) **American Airlines**, a unit of AMR, immediately matched the move,
 spokesman **Tim Wagner** said

we would learn rich conjunction features like this one:

M1 = ORG & M2 = PER & nextword="said"& path= $NP \uparrow NP \uparrow S \uparrow S \downarrow NP$

The result is a supervised classifier that has a huge rich set of features to use in detecting relations. Since not every test sentence will have one of the training

relations, the classifier will also need to be able to label an example as *no-relation*. This label is trained by randomly selecting entity pairs that do not appear in any Freebase relation, extracting features for them, and building a feature vector for each such tuple. The final algorithm is sketched in Fig. 18.16.

```
function DISTANT SUPERVISION(Database D, Text T) returns relation classifier C
foreach relation R
    foreach tuple (e1,e2) of entities with relation R in D
        sentences  $\leftarrow$  Sentences in T that contain e1 and e2
        f  $\leftarrow$  Frequent features in sentences
        observations  $\leftarrow$  observations + new training tuple (e1, e2, f, R)
    C  $\leftarrow$  Train supervised classifier on observations
    return C
```

Figure 18.16 The distant supervision algorithm for relation extraction. A neural classifier might not need to use the feature set *f*.

Distant supervision shares advantages with each of the methods we've examined. Like supervised classification, distant supervision uses a classifier with lots of features, and supervised by detailed hand-created knowledge. Like pattern-based classifiers, it can make use of high-precision evidence for the relation between entities. Indeed, distance supervision systems learn patterns just like the hand-built patterns of early relation extractors. For example the *is-a* or *hypernym* extraction system of Snow et al. (2005) used hypernym/hyponym NP pairs from WordNet as distant supervision, and then learned new patterns from large amounts of text. Their system induced exactly the original 5 template patterns of Hearst (1992a), but also 70,000 additional patterns including these four:

NP_H like NP	<i>Many hormones like leptin...</i>
NP_H called NP	<i>...using a markup language called XHTML</i>
NP is a NP_H	<i>Ruby is a programming language...</i>
NP , a NP_H	<i>IBM, a company with a long...</i>

This ability to use a large number of features simultaneously means that, unlike the iterative expansion of patterns in seed-based systems, there's no semantic drift. Like unsupervised classification, it doesn't use a labeled training corpus of texts, so it isn't sensitive to genre issues in the training corpus, and relies on very large amounts of unlabeled data. Distant supervision also has the advantage that it can create training tuples to be used with neural classifiers, where features are not required.

But distant supervision can only help in extracting relations for which a large enough database already exists. To extract new relations without datasets, or relations for new domains, purely unsupervised methods must be used.

18.2.5 Unsupervised Relation Extraction

open
information
extraction

The goal of unsupervised relation extraction is to extract relations from the web when we have no labeled training data, and not even any list of relations. This task is often called **open information extraction** or **Open IE**. In Open IE, the relations are simply strings of words (usually beginning with a verb).

For example, the **ReVerb** system (Fader et al., 2011) extracts a relation from a sentence *s* in 4 steps:

1. Run a part-of-speech tagger and entity chunker over s
2. For each verb in s , find the longest sequence of words w that start with a verb and satisfy syntactic and lexical constraints, merging adjacent matches.
3. For each phrase w , find the nearest noun phrase x to the left which is not a relative pronoun, wh-word or existential “there”. Find the nearest noun phrase y to the right.
4. Assign confidence c to the relation $r = (x, w, y)$ using a confidence classifier and return it.

A relation is only accepted if it meets syntactic and lexical constraints. The syntactic constraints ensure that it is a verb-initial sequence that might also include nouns (relations that begin with light verbs like *make*, *have*, or *do* often express the core of the relation with a noun, like *have a hub in*):

```
V | VP | VW*P
V = verb particle? adv?
W = (noun | adj | adv | pron | det )
P = (prep | particle | inf. marker)
```

The lexical constraints are based on a dictionary D that is used to prune very rare, long relation strings. The intuition is to eliminate candidate relations that don't occur with sufficient number of distinct argument types and so are likely to be bad examples. The system first runs the above relation extraction algorithm offline on 500 million web sentences and extracts a list of all the relations that occur after normalizing them (removing inflection, auxiliary verbs, adjectives, and adverbs). Each relation r is added to the dictionary if it occurs with at least 20 different arguments. Fader et al. (2011) used a dictionary of 1.7 million normalized relations.

Finally, a confidence value is computed for each relation using a logistic regression classifier. The classifier is trained by taking 1000 random web sentences, running the extractor, and hand labelling each extracted relation as correct or incorrect. A confidence classifier is then trained on this hand-labeled data, using features of the relation and the surrounding words. Fig. 18.17 shows some sample features used in the classification.

```
(x,r,y) covers all words in s
the last preposition in r is for
the last preposition in r is on
len(s) ≤ 10
there is a coordinating conjunction to the left of r in s
r matches a lone V in the syntactic constraints
there is preposition to the left of x in s
there is an NP to the right of y in s
```

Figure 18.17 Features for the classifier that assigns confidence to relations extracted by the Open Information Extraction system REVERB (Fader et al., 2011).

For example the following sentence:

- (18.13) United has a hub in Chicago, which is the headquarters of United
Continental Holdings.

has the relation phrases *has a hub in* and *is the headquarters of* (it also has *has* and *is*, but longer phrases are preferred). Step 3 finds *United* to the left and *Chicago* to the right of *has a hub in*, and skips over *which* to find *Chicago* to the left of *is the headquarters of*. The final output is:

```
r1: <United, has a hub in, Chicago>
r2: <Chicago, is the headquarters of, United Continental Holdings>
```

The great advantage of unsupervised relation extraction is its ability to handle a huge number of relations without having to specify them in advance. The disadvantage is the need to map these large sets of strings into some canonical form for adding to databases or other knowledge sources. Current methods focus heavily on relations expressed with verbs, and so will miss many relations that are expressed nominally.

18.2.6 Evaluation of Relation Extraction

Supervised relation extraction systems are evaluated by using test sets with human-annotated, gold-standard relations and computing precision, recall, and F-measure. Labeled precision and recall require the system to classify the relation correctly, whereas unlabeled methods simply measure a system's ability to detect entities that are related.

Semi-supervised and **unsupervised** methods are much more difficult to evaluate, since they extract totally new relations from the web or a large text. Because these methods use very large amounts of text, it is generally not possible to run them solely on a small labeled test set, and as a result it's not possible to pre-annotate a gold set of correct instances of relations.

For these methods it's possible to approximate (only) precision by drawing a random sample of relations from the output, and having a human check the accuracy of each of these relations. Usually this approach focuses on the **tuples** to be extracted from a body of text rather than on the relation **mentions**; systems need not detect every mention of a relation to be scored correctly. Instead, the evaluation is based on the set of tuples occupying the database when the system is finished. That is, we want to know if the system can discover that Ryanair has a hub at Charleroi; we don't really care how many times it discovers it. The estimated precision \hat{P} is then

$$\hat{P} = \frac{\text{\# of correctly extracted relation tuples in the sample}}{\text{total \# of extracted relation tuples in the sample.}} \quad (18.14)$$

Another approach that gives us a little bit of information about recall is to compute precision at different levels of recall. Assuming that our system is able to rank the relations it produces (by probability, or confidence) we can separately compute precision for the top 1000 new relations, the top 10,000 new relations, the top 100,000, and so on. In each case we take a random sample of that set. This will show us how the precision curve behaves as we extract more and more tuples. But there is no way to directly evaluate recall.

18.3 Extracting Times

Times and dates are a particularly important kind of named entity that play a role in question answering, in calendar and personal assistant applications. In order to reason about times and dates, after we extract these **temporal expressions** they must be **normalized**—converted to a standard format so we can reason about them. In this section we consider both the extraction and normalization of temporal expressions.

18.3.1 Temporal Expression Extraction

absolute Temporal expressions are those that refer to absolute points in time, relative times, durations, and sets of these. **Absolute** temporal expressions are those that can be mapped directly to calendar dates, times of day, or both. **Relative** temporal expressions map to particular times through some other reference point (as in *a week from last Tuesday*). Finally, **durations** denote spans of time at varying levels of granularity (seconds, minutes, days, weeks, centuries, etc.). Figure 18.18 lists some sample temporal expressions in each of these categories.

Absolute	Relative	Durations
April 24, 1916	yesterday	four hours
The summer of '77	next semester	three weeks
10:15 AM	two weeks from yesterday	six days
The 3rd quarter of 2006	last quarter	the last three quarters

Figure 18.18 Examples of absolute, relational and durational temporal expressions.

lexical triggers Temporal expressions are grammatical constructions that have temporal **lexical triggers** as their heads. Lexical triggers might be nouns, proper nouns, adjectives, and adverbs; full temporal expressions consist of their phrasal projections: noun phrases, adjective phrases, and adverbial phrases. Figure 18.19 provides examples.

Category	Examples
Noun	<i>morning, noon, night, winter, dusk, dawn</i>
Proper Noun	<i>January, Monday, Ides, Easter, Rosh Hashana, Ramadan, Tet</i>
Adjective	<i>recent, past, annual, former</i>
Adverb	<i>hourly, daily, monthly, yearly</i>

Figure 18.19 Examples of temporal lexical triggers.

Let's look at the TimeML annotation scheme, in which temporal expressions are annotated with an XML tag, **TIMEX3**, and various attributes to that tag (Pustejovsky et al. 2005, Ferro et al. 2005). The following example illustrates the basic use of this scheme (we defer discussion of the attributes until Section 18.3.2).

A fare increase initiated <TIMEX3>last week</TIMEX3> by UAL Corp's United Airlines was matched by competitors over <TIMEX3>the weekend</TIMEX3>, marking the second successful fare increase in <TIMEX3>two weeks</TIMEX3>.

The temporal expression recognition task consists of finding the start and end of all of the text spans that correspond to such temporal expressions. **Rule-based approaches** to temporal expression recognition use cascades of automata to recognize patterns at increasing levels of complexity. Tokens are first part-of-speech tagged, and then larger and larger chunks are recognized from the results from previous stages, based on patterns containing trigger words (e.g., *February*) or classes (e.g., *MONTH*). Figure 18.20 gives a fragment from a rule-based system.

Sequence-labeling approaches follow the same IOB scheme used for named-entity tags, marking words that are either inside, outside or at the beginning of a TIMEX3-delimited temporal expression with the I, O, and B tags as follows:

A fare increase initiated last week by UAL Corp's...

O O O O B I O O O

```
# yesterday/today/tomorrow
$string =~ s/((($OT+$the$CT+\s+)?$OT+day$CT+\s+$OT+(before|after)$CT+\s+)?$OT+$TRElDayExpr$CT+
(\s+$OT+($morning|afternoon|evening|night)$CT+)?)/<TIME$tever TYPE="DATE">$1
</TIME$tever>/gso;

$string =~ s/($OT+\w+$CT+\s+)<TIME$tever TYPE="DATE" [^>]*>($OT+(Today|Tonight)$CT+)
</TIME$tever>/${1$4}/gso;

# this (morning/afternoon/evening)
$string =~ s/((($OT+($early|late)$CT+\s+)?$OT+this$CT+\s*$OT+($morning|afternoon|evening)$CT+)/
<TIME$tever TYPE="DATE">$1</TIME$tever>/gso;
$string =~ s/((($OT+($early|late)$CT+\s+)?$OT+last$CT+\s*$OT+night$CT+)?<TIME$tever
TYPE="DATE">$1</TIME$tever>/gso;
```

Figure 18.20 Perl fragment from the GUTime temporal tagging system in Tarsqi (Verhagen et al., 2005).

Features are extracted from the token and its context, and a statistical sequence labeler is trained (any sequence model can be used). Figure 18.21 lists standard features used in temporal tagging.

Feature	Explanation
Token	The target token to be labeled
Tokens in window	Bag of tokens in the window around a target
Shape	Character shape features
POS	Parts of speech of target and window words
Chunk tags	Base-phrase chunk tag for target and words in a window
Lexical triggers	Presence in a list of temporal terms

Figure 18.21 Typical features used to train IOB-style temporal expression taggers.

Temporal expression recognizers are evaluated with the usual recall, precision, and F -measures. A major difficulty for all of these very lexicalized approaches is avoiding expressions that trigger false positives:

(18.15) 1984 tells the story of Winston Smith...

(18.16) ...U2's classic *Sunday Bloody Sunday*

18.3.2 Temporal Normalization

temporal normalization

Temporal normalization is the process of mapping a temporal expression to either a specific point in time or to a duration. Points in time correspond to calendar dates, to times of day, or both. Durations primarily consist of lengths of time but may also include information about start and end points. Normalized times are represented with the VALUE attribute from the ISO 8601 standard for encoding temporal values (ISO8601, 2004). Fig. 18.22 reproduces our earlier example with the value attributes added in.

```
<TIME3 id="t1" type="DATE" value="2007-07-02" functionInDocument="CREATION_TIME"
> July 2, 2007 </TIME3> A fare increase initiated <TIME3 id="t2" type="DATE"
value="2007-W26" anchorTimeID="t1">last week</TIME3> by United Airlines was
matched by competitors over <TIME3 id="t3" type="DURATION" value="P1WE"
anchorTimeID="t1"> the weekend </TIME3>, marking the second successful fare
increase in <TIME3 id="t4" type="DURATION" value="P2W" anchorTimeID="t1"> two
weeks </TIME3>.
```

Figure 18.22 TimeML markup including normalized values for temporal expressions.

The dateline, or document date, for this text was *July 2, 2007*. The ISO representation for this kind of expression is YYYY-MM-DD, or in this case, 2007-07-02.

The encodings for the temporal expressions in our sample text all follow from this date, and are shown here as values for the VALUE attribute.

The first temporal expression in the text proper refers to a particular week of the year. In the ISO standard, weeks are numbered from 01 to 53, with the first week of the year being the one that has the first Thursday of the year. These weeks are represented with the template YYYY-Wnn. The ISO week for our document date is week 27; thus the value for *last week* is represented as “2007-W26”.

The next temporal expression is *the weekend*. ISO weeks begin on Monday; thus, weekends occur at the end of a week and are fully contained within a single week. Weekends are treated as durations, so the value of the VALUE attribute has to be a length. Durations are represented according to the pattern Pnx, where n is an integer denoting the length and x represents the unit, as in P3Y for *three years* or P2D for *two days*. In this example, one weekend is captured as P1WE. In this case, there is also sufficient information to anchor this particular weekend as part of a particular week. Such information is encoded in the ANCHORTIMEID attribute. Finally, the phrase *two weeks* also denotes a duration captured as P2W. There is a lot more to the various temporal annotation standards—far too much to cover here. Figure 18.23 describes some of the basic ways that other times and durations are represented. Consult ISO8601 (2004), Ferro et al. (2005), and Pustejovsky et al. (2005) for more details.

Unit	Pattern	Sample Value
Fully specified dates	YYYY-MM-DD	1991-09-28
Weeks	YYYY-Wnn	2007-W27
Weekends	PnWE	P1WE
24-hour clock times	HH:MM:SS	11:13:45
Dates and times	YYYY-MM-DDTHH:MM:SS	1991-09-28T11:00:00
Financial quarters	Qn	1999-Q3

Figure 18.23 Sample ISO patterns for representing various times and durations.

Most current approaches to temporal normalization are rule-based (Chang and Manning 2012, Strötgen and Gertz 2013). Patterns that match temporal expressions are associated with semantic analysis procedures. As in the compositional rule-to-rule approach introduced in Chapter 17, the meaning of a constituent is computed from the meaning of its parts using a method specific to the constituent, although here the semantic composition rules involve temporal arithmetic rather than λ -calculus attachments.

fully qualified

Fully qualified date expressions contain a year, month, and day in some conventional form. The units in the expression must be detected and then placed in the correct place in the corresponding ISO pattern. The following pattern normalizes expressions like *April 24, 1916*.

$$FQTE \rightarrow Month\ Date\ , Year \quad \{Year.val - Month.val - Date.val\}$$

The non-terminals *Month*, *Date*, and *Year* represent constituents that have already been recognized and assigned semantic values, accessed through the **.val* notation. The value of this *FQE* constituent can, in turn, be accessed as *FQTE.val* during further processing.

Fully qualified temporal expressions are fairly rare in real texts. Most temporal expressions in news articles are incomplete and are only implicitly anchored, often with respect to the dateline of the article, which we refer to as the document’s

temporal anchor

temporal anchor. The values of temporal expressions such as *today*, *yesterday*, or *tomorrow* can all be computed with respect to this temporal anchor. The semantic procedure for *today* simply assigns the anchor, and the attachments for *tomorrow* and *yesterday* add a day and subtract a day from the anchor, respectively. Of course, given the cyclic nature of our representations for months, weeks, days, and times of day, our temporal arithmetic procedures must use modulo arithmetic appropriate to the time unit being used.

Unfortunately, even simple expressions such as *the weekend* or *Wednesday* introduce a fair amount of complexity. In our current example, *the weekend* clearly refers to the weekend of the week that immediately precedes the document date. But this won't always be the case, as is illustrated in the following example.

- (18.17) Random security checks that began yesterday at Sky Harbor will continue at least through the weekend.

In this case, the expression *the weekend* refers to the weekend of the week that the anchoring date is part of (i.e., the coming weekend). The information that signals this meaning comes from the tense of *continue*, the verb governing *the weekend*.

Relative temporal expressions are handled with temporal arithmetic similar to that used for *today* and *yesterday*. The document date indicates that our example article is ISO week 27, so the expression *last week* normalizes to the current week minus 1. To resolve ambiguous *next* and *last* expressions we consider the distance from the anchoring date to the nearest unit. *Next Friday* can refer either to the immediately next Friday or to the Friday following that, but the closer the document date is to a Friday, the more likely it is that the phrase will skip the nearest one. Such ambiguities are handled by encoding language and domain-specific heuristics into the temporal attachments.

18.4 Extracting Events and their Times

event extraction

The task of **event extraction** is to identify mentions of events in texts. For the purposes of this task, an event mention is any expression denoting an event or state that can be assigned to a particular point, or interval, in time. The following markup of the sample text on page 343 shows all the events in this text.

[EVENT Citing] high fuel prices, United Airlines [EVENT said] Friday it has [EVENT increased] fares by \$6 per round trip on flights to some cities also served by lower-cost carriers. American Airlines, a unit of AMR Corp., immediately [EVENT matched] [EVENT the move], spokesman Tim Wagner [EVENT said]. United, a unit of UAL Corp., [EVENT said] [EVENT the increase] took effect Thursday and [EVENT applies] to most routes where it [EVENT competes] against discount carriers, such as Chicago to Dallas and Denver to San Francisco.

In English, most event mentions correspond to verbs, and most verbs introduce events. However, as we can see from our example, this is not always the case. Events can be introduced by noun phrases, as in *the move* and *the increase*, and some verbs fail to introduce events, as in the phrasal verb *took effect*, which refers to when the event began rather than to the event itself. Similarly, light verbs such as *make*, *take*, and *have* often fail to denote events; for light verbs the event is often expressed by the nominal direct object (*took a flight*), and these light verbs just provide a syntactic structure for the noun's arguments.

reporting events

Various versions of the event extraction task exist, depending on the goal. For example in the TempEval shared tasks ([Verhagen et al. 2009](#)) the goal is to extract events and aspects like their aspectual and temporal properties. Events are to be classified as actions, states, **reporting events** (*say, report, tell, explain*), perception events, and so on. The aspect, tense, and modality of each event also needs to be extracted. Thus for example the various *said* events in the sample text would be annotated as (class=REPORTING, tense=PAST, aspect=PERFECTIVE).

Event extraction is generally modeled via supervised learning, detecting events via sequence models with IOB tagging, and assigning event classes and attributes with multi-class classifiers. Feature-based models use surface information like parts of speech, lexical items, and verb tense information; see Fig. 18.24.

Feature	Explanation
Character affixes	Character-level prefixes and suffixes of target word
Nominalization suffix	Character-level suffixes for nominalizations (e.g., <i>-tion</i>)
Part of speech	Part of speech of the target word
Light verb	Binary feature indicating that the target is governed by a light verb
Subject syntactic category	Syntactic category of the subject of the sentence
Morphological stem	Stemmed version of the target word
Verb root	Root form of the verb basis for a nominalization
WordNet hypernyms	Hypernym set for the target

Figure 18.24 Features commonly used in both rule-based and machine learning approaches to event detection.

Allen relations**TimeBank**

18.4.1 Temporal Ordering of Events

With both the events and the temporal expressions in a text having been detected, the next logical task is to use this information to fit the events into a complete timeline. Such a timeline would be useful for applications such as question answering and summarization. This ambitious task is the subject of considerable current research but is beyond the capabilities of current systems.

A somewhat simpler, but still useful, task is to impose a partial ordering on the events and temporal expressions mentioned in a text. Such an ordering can provide many of the same benefits as a true timeline. An example of such a partial ordering is the determination that the fare increase by *American Airlines* came *after* the fare increase by *United* in our sample text. Determining such an ordering can be viewed as a binary relation detection and classification task similar to those described earlier in Section 18.2. The temporal relation between events is classified into one of the standard set of **Allen relations** shown in Fig. 18.25 ([Allen, 1984](#)), using feature-based classifiers as in Section 18.2, trained on the TimeBank corpus with features like words/embeddings, parse paths, tense and aspect.

The **TimeBank** corpus consists of text annotated with much of the information we've been discussing throughout this section ([Pustejovsky et al., 2003b](#)). TimeBank 1.2 consists of 183 news articles selected from a variety of sources, including the Penn TreeBank and PropBank collections.

Each article in the TimeBank corpus has had the temporal expressions and event mentions in them explicitly annotated in the TimeML annotation ([Pustejovsky et al., 2003a](#)). In addition to temporal expressions and events, the TimeML annotation provides temporal links between events and temporal expressions that specify the nature of the relation between them. Consider the following sample sentence and

**Figure 18.25** The 13 temporal relations from Allen (1984).

```
<TIMEX3 tid="t57" type="DATE" value="1989-10-26" functionInDocument="CREATION_TIME">
10/26/89 </TIMEX3>
```

Delta Air Lines earnings <EVENT eid="e1" class="OCCURRENCE"> soared </EVENT> 33% to a record in <TIMEX3 tid="t58" type="DATE" value="1989-Q1" anchorTimeID="t57"> the fiscal first quarter </TIMEX3>, <EVENT eid="e3" class="OCCURRENCE">bucking</EVENT> the industry trend toward <EVENT eid="e4" class="OCCURRENCE">declining</EVENT> profits.

Figure 18.26 Example from the TimeBank corpus.

its corresponding markup shown in Fig. 18.26, selected from one of the TimeBank documents.

(18.18) Delta Air Lines earnings soared 33% to a record in the fiscal first quarter, bucking the industry trend toward declining profits.

As annotated, this text includes three events and two temporal expressions. The events are all in the occurrence class and are given unique identifiers for use in further annotations. The temporal expressions include the creation time of the article, which serves as the document time, and a single temporal expression within the text.

In addition to these annotations, TimeBank provides four links that capture the temporal relations between the events and times in the text, using the Allen relations from Fig. 18.25. The following are the within-sentence temporal relations annotated for this example.

- Soaring_{e1} is **included** in the fiscal first quarter_{t58}
- Soaring_{e1} is **before** 1989-10-26_{t57}
- Soaring_{e1} is **simultaneous** with the bucking_{e3}
- Declining_{e4} **includes** soaring_{e1}

18.5 Template Filling

scripts Many texts contain reports of events, and possibly sequences of events, that often correspond to fairly common, stereotypical situations in the world. These abstract situations or stories, related to what have been called **scripts** (Schank and Abelson, 1977), consist of prototypical sequences of sub-events, participants, and their roles. The strong expectations provided by these scripts can facilitate the proper classification of entities, the assignment of entities into roles and relations, and most critically, the drawing of inferences that fill in things that have been left unsaid. In their simplest form, such scripts can be represented as **templates** consisting of fixed sets of **slots** that take as values **slot-fillers** belonging to particular classes. The task of **template filling** is to find documents that invoke particular scripts and then fill the slots in the associated templates with fillers extracted from the text. These slot-fillers may consist of text segments extracted directly from the text, or they may consist of concepts that have been inferred from text elements through some additional processing.

templates

template filling

A filled template from our original airline story might look like the following.

FARE-RAISE ATTEMPT:	[LEAD AIRLINE:	UNITED AIRLINES]
		AMOUNT:	\$6	
		EFFECTIVE DATE:	2006-10-26	
		FOLLOWER:	AMERICAN AIRLINES	

This template has four slots (LEAD AIRLINE, AMOUNT, EFFECTIVE DATE, FOLLOWER). The next section describes a standard sequence-labeling approach to filling slots. Section 18.5.2 then describes an older system based on the use of cascades of finite-state transducers and designed to address a more complex template-filling task that current learning-based systems don't yet address.

18.5.1 Machine Learning Approaches to Template Filling

In the standard paradigm for template filling, we are given training documents with text spans annotated with pre-defined templates and their slot fillers. Our goal is to create one template for each event in the input, filling in the slots with text spans.

template recognition

The task is generally modeled by training two separate supervised systems. The first system decides whether the template is present in a particular sentence. This task is called **template recognition** or sometimes, in a perhaps confusing bit of terminology, *event recognition*. Template recognition can be treated as a text classification task, with features extracted from every sequence of words that was labeled in training documents as filling any slot from the template being detected. The usual set of features can be used: tokens, embeddings, word shapes, part-of-speech tags, syntactic chunk tags, and named entity tags.

role-filler extraction

The second system has the job of **role-filler extraction**. A separate classifier is trained to detect each role (LEAD-AIRLINE, AMOUNT, and so on). This can be a

binary classifier that is run on every noun-phrase in the parsed input sentence, or a sequence model run over sequences of words. Each role classifier is trained on the labeled data in the training set. Again, the usual set of features can be used, but now trained only on an individual noun phrase or the fillers of a single slot.

Multiple non-identical text segments might be labeled with the same slot label. For example in our sample text, the strings *United* or *United Airlines* might be labeled as the LEAD AIRLINE. These are not incompatible choices and the coreference resolution techniques introduced in Chapter 22 can provide a path to a solution.

A variety of annotated collections have been used to evaluate this style of approach to template filling, including sets of job announcements, conference calls for papers, restaurant guides, and biological texts. Recent work focuses on extracting templates in cases where there is no training data or even predefined templates, by inducing templates as sets of linked events (Chambers and Jurafsky, 2011).

18.5.2 Earlier Finite-State Template-Filling Systems

The templates above are relatively simple. But consider the task of producing a template that contained all the information in a text like this one (Grishman and Sundheim, 1995):

Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and “metal wood” clubs a month.

The MUC-5 ‘joint venture’ task (the *Message Understanding Conferences* were a series of U.S. government-organized information-extraction evaluations) was to produce hierarchically linked templates describing joint ventures. Figure 18.27 shows a structure produced by the FASTUS system (Hobbs et al., 1997). Note how the filler of the ACTIVITY slot of the TIE-UP template is itself a template with slots.

Tie-up-1		Activity-1:		
RELATIONSHIP	tie-up	COMPANY	Bridgestone Sports Taiwan Co.	
ENTITIES	Bridgestone Sports Co.	PRODUCT	iron and “metal wood” clubs	
	a local concern	START DATE	DURING: January 1990	
	a Japanese trading house			
JOINT VENTURE	Bridgestone Sports Taiwan Co.			
ACTIVITY	Activity-1			
AMOUNT	NT\$20000000			

Figure 18.27 The templates produced by FASTUS given the input text on page 350.

Early systems for dealing with these complex templates were based on cascades of transducers based on handwritten rules, as sketched in Fig. 18.28.

The first four stages use handwritten regular expression and grammar rules to do basic tokenization, chunking, and parsing. Stage 5 then recognizes entities and events with a FST-based recognizer and inserts the recognized objects into the appropriate slots in templates. This FST recognizer is based on hand-built regular expressions like the following (NG indicates Noun-Group and VG Verb-Group), which matches the first sentence of the news story above.

```
NG(Company/ies) VG(Set-up) NG(Joint-Venture) with NG(Company/ies)
VG(Produce) NG(Product)
```

No.	Step	Description
1	Tokens	Tokenize input stream of characters
2	Complex Words	Multiword phrases, numbers, and proper names.
3	Basic phrases	Segment sentences into noun and verb groups
4	Complex phrases	Identify complex noun groups and verb groups
5	Semantic Patterns	Identify entities and events, insert into templates.
6	Merging	Merge references to the same entity or event

Figure 18.28 Levels of processing in FASTUS (Hobbs et al., 1997). Each level extracts a specific type of information which is then passed on to the next higher level.

The result of processing these two sentences is the five draft templates (Fig. 18.29) that must then be merged into the single hierarchical structure shown in Fig. 18.27. The merging algorithm, after performing coreference resolution, merges two activities that are likely to be describing the same events.

#	Template/Slot	Value
1	RELATIONSHIP:	TIE-UP
	ENTITIES:	Bridgestone Co., a local concern, a Japanese trading house
2	ACTIVITY:	PRODUCTION
	PRODUCT:	“golf clubs”
3	RELATIONSHIP:	TIE-UP
	JOINT VENTURE:	“Bridgestone Sports Taiwan Co.”
	AMOUNT:	NT\$20000000
4	ACTIVITY:	PRODUCTION
	COMPANY:	“Bridgestone Sports Taiwan Co.”
	STARTDATE:	DURING: January 1990
5	ACTIVITY:	PRODUCTION
	PRODUCT:	“iron and “metal wood” clubs”

Figure 18.29 The five partial templates produced by stage 5 of FASTUS. These templates are merged in stage 6 to produce the final template shown in Fig. 18.27 on page 350.

18.6 Summary

This chapter has explored techniques for extracting limited forms of semantic content from texts.

- **Named entities** can be recognized and classified by featured-based or neural sequence labeling techniques.
- **Relations among entities** can be extracted by pattern-based approaches, supervised learning methods when annotated training data is available, lightly supervised **bootstrapping** methods when small numbers of **seed tuples** or **seed patterns** are available, **distant supervision** when a database of relations is available, and **unsupervised** or **Open IE** methods.
- Reasoning about time can be facilitated by detection and normalization of **temporal expressions** through a combination of statistical learning and rule-based methods.
- **Events** can be detected and ordered in time using sequence models and classifiers trained on temporally- and event-labeled data like the **TimeBank corpus**.

- **Template-filling** applications can recognize stereotypical situations in texts and assign elements from the text to roles represented as **fixed sets of slots**.

Bibliographical and Historical Notes

The earliest work on information extraction addressed the template-filling task in the context of the Frump system (DeJong, 1982). Later work was stimulated by the U.S. government-sponsored MUC conferences (Sundheim 1991, Sundheim 1992, Sundheim 1993, Sundheim 1995). Early MUC systems like CIRCUS system (Lehnert et al., 1991) and SCISOR (Jacobs and Rau, 1990) were quite influential and inspired later systems like FASTUS (Hobbs et al., 1997). Chinchor et al. (1993) describe the MUC evaluation techniques.

Due to the difficulty of porting systems from one domain to another, attention shifted to machine learning approaches.

Early supervised learning approaches to IE (Cardie 1993, Cardie 1994, Riloff 1993, Soderland et al. 1995, Huffman 1996) focused on automating the knowledge acquisition process, mainly for finite-state rule-based systems. Their success, and the earlier success of HMM-based speech recognition, led to the use of sequence labeling (HMMs: Bikel et al. 1997; MEMMs McCallum et al. 2000; CRFs: Lafferty et al. 2001), and a wide exploration of features (Zhou et al., 2005). Neural approaches to NER mainly follow from the pioneering results of Collobert et al. (2011), who applied a CRF on top of a convolutional net. BiLSTMs with word and character-based embeddings as input followed shortly and became a standard neural algorithm for NER (Huang et al. 2015, Ma and Hovy 2016, Lample et al. 2016b).

Neural algorithms for relation extraction often explore architectures that can handle entities far apart in the sentence: recursive networks (Socher et al., 2012), convolutional nets (dos Santos et al., 2015), or chain or tree LSTMS (Miwa and Bansal 2016, Peng et al. 2017).

KBP
slot filling

Progress in this area continues to be stimulated by formal evaluations with shared benchmark datasets, including the Automatic Content Extraction (ACE) evaluations of 2000-2007 on named entity recognition, relation extraction, and temporal expressions³, the **KBP (Knowledge Base Population)** evaluations (Ji et al. 2010, Surdeanu 2013) of relation extraction tasks like **slot filling** (extracting attributes ('slots') like age, birthplace, and spouse for a given entity) and a series of SemEval workshops (Hendrickx et al., 2009).

Semisupervised relation extraction was first proposed by Hearst (1992b), and extended by systems like AutoSlog-TS (Riloff, 1996), DIPRE (Brin, 1998), SNOWBALL (Agichtein and Gravano, 2000), and (Jones et al., 1999). The distant supervision algorithm we describe was drawn from Mintz et al. (2009), who coined the term 'distant supervision', but similar ideas occurred in earlier systems like Craven and Kumlien (1999) and Morgan et al. (2004) under the name *weakly labeled data*, as well as in Snow et al. (2005) and Wu and Weld (2007). Among the many extensions are Wu and Weld (2010), Riedel et al. (2010), and Ritter et al. (2013). Open IE systems include KNOWITALL Etzioni et al. (2005), TextRunner (Banko et al., 2007), and REVERB (Fader et al., 2011). See Riedel et al. (2013) for a universal schema that combines the advantages of distant supervision and Open IE.

³ www.nist.gov/speech/tests/ace/

HeidelTime ([Strötgen and Gertz, 2013](#)) and SUTime ([Chang and Manning, 2012](#)) are downloadable temporal extraction and normalization systems. The 2013 TempEval challenge is described in [UzZaman et al. \(2013\)](#); [Chambers \(2013\)](#) and [Bethard \(2013\)](#) give typical approaches.

Exercises

- 18.1** Develop a set of regular expressions to recognize the character shape features described on page [329](#).
- 18.2** The IOB labeling scheme given in this chapter isn't the only possible one. For example, an E tag might be added to mark the end of entities, or the B tag can be reserved only for those situations where an ambiguity exists between adjacent entities. Propose a new set of IOB tags for use with your NER system. Experiment with it and compare its performance with the scheme presented in this chapter.
- 18.3** Names of works of art (books, movies, video games, etc.) are quite different from the kinds of named entities we've discussed in this chapter. Collect a list of names of works of art from a particular category from a Web-based source (e.g., [gutenberg.org](#), [amazon.com](#), [imdb.com](#), etc.). Analyze your list and give examples of ways that the names in it are likely to be problematic for the techniques described in this chapter.
- 18.4** Develop an NER system specific to the category of names that you collected in the last exercise. Evaluate your system on a collection of text likely to contain instances of these named entities.
- 18.5** Acronym expansion, the process of associating a phrase with an acronym, can be accomplished by a simple form of relational analysis. Develop a system based on the relation analysis approaches described in this chapter to populate a database of acronym expansions. If you focus on English **Three Letter Acronyms** (TLAs) you can evaluate your system's performance by comparing it to Wikipedia's TLA page.
- 18.6** A useful functionality in newer email and calendar applications is the ability to associate temporal expressions connected with events in email (doctor's appointments, meeting planning, party invitations, etc.) with specific calendar entries. Collect a corpus of email containing temporal expressions related to event planning. How do these expressions compare to the kinds of expressions commonly found in news text that we've been discussing in this chapter?
- 18.7** Acquire the CMU seminar corpus and develop a template-filling system by using any of the techniques mentioned in Section [18.5](#). Analyze how well your system performs as compared with state-of-the-art results on this corpus.

CHAPTER

19

Word Senses and WordNet

*To get a single right meaning is better than a ship-load of pearls,
To resolve a single doubt is like the bottom falling off the bucket.*

Yuen Mei 袁枚(1785) (translation by Arthur Waley)

ambiguous

Words are **ambiguous**: the same word can be used to mean different things. For example in Chapter 6 we saw that the word “mouse” has (at least) two meanings: (1) a small rodent, or (2) a hand-operated device to control a cursor. Or the word “bank” can mean: (1) a financial institution or (2) a sloping mound.

word sense

We say that the words ‘mouse’ or ‘bank’ are **polysemous** (from Greek ‘having many senses’, *poly-* ‘many’ + *sema*, ‘sign, mark’).¹ A **sense** (or **word sense**) is a discrete representation of one aspect of the meaning of a word. In this chapter we discuss word senses in more detail and introduce **WordNet**, a large online **thesaurus** —a database that represents word senses—with versions in many languages. WordNet also represents relations between senses. For example, there is an **IS-A** relation between *dog* and *mammal* (a dog is a kind of mammal) and a **part-whole** relation between *engine* and *car* (an engine is a part of a car).

WordNet

Knowing the relation between two senses can play an important role in language understanding. Consider the **antonymy** relation. Two words are antonyms if they have opposite meanings, like *long* and *short*, or *up* and *down*. Distinguishing these is quite important for language understanding (if a user asks a dialogue agent to turn up the music, it would be unfortunate to instead turn it down). But in fact in embedding models like word2vec, antonyms are easily confused with each other, because often one of the closest words in embedding space to a word (e.g., *up*) is its antonym (e.g., *down*). Thesauruses that represent this relationship can help!

**word sense
disambiguation**

We also introduce **word sense disambiguation (WSD)**, the task of determining which sense of a word is being used in a particular context. We’ll give supervised and unsupervised algorithms for deciding which sense was intended in a particular context. This task has a very long history in computational linguistics and many applications. In question answering, we can be more helpful to a user who asks about “bat care” if we know which sense of bat is relevant. (Is the user a vampire? or just wants to play baseball.) And the different senses of a word often have different translations; in Spanish the animal bat is a *murciélagos* while the baseball bat is a *bate*, and indeed word sense algorithms may help improve MT (Pu et al., 2018). Finally, WSD has long been used as a tool for evaluating natural language understanding models, and understanding how models represent different word senses is an important analytic direction.

¹ You may also see the word **polysemy** used in a different way, to refer only to cases where a word’s senses have some sort of semantic relation, and use the word **homonymy** for cases with no relation between the senses.

19.1 Word Senses

word sense A **sense** (or **word sense**) is a discrete representation of one aspect of the meaning of a word. Loosely following lexicographic tradition, we represent each sense with a superscript: **bank¹** and **bank²**, **mouse¹** and **mouse²**. In context, it's easy to see the different meanings:

mouse¹ : a *mouse* controlling a computer system in 1968.

mouse² : a quiet animal like a *mouse*

bank¹ : ...a *bank* can hold the investments in a custodial account ...

bank² : ...as agriculture burgeons on the east *bank*, the river ...

19.1.1 Defining Word Senses

How can we define the meaning of a word sense? We introduced in Chapter 6 the standard computational approach of representing a word as an **embedding**, a point in semantic space. The intuition of embedding models like word2vec or GloVe is that the meaning of a word can be defined by its co-occurrences, the counts of words that often occur nearby. But that doesn't tell us how to define the meaning of a word *sense*. Contextual embeddings like ELMo or BERT go further by offering an embedding that represents the meaning of a word in its textual context, and we'll see that contextual embeddings lie at the heart of modern algorithms for word sense disambiguation.

But first, we need to consider the alternative ways that dictionaries and thesauruses offer for defining senses. One is based on the fact that dictionaries or thesauruses give textual definitions for each sense called **glosses**. Here are the glosses for two senses of *bank*:

1. financial institution that accepts deposits and channels the money into lending activities
2. sloping land (especially the slope beside a body of water)

Glosses are not a formal meaning representation; they are just written for people. Consider the following fragments from the definitions of *right*, *left*, *red*, and *blood* from the *American Heritage Dictionary* (Morris, 1985).

right	<i>adj.</i> located nearer the right hand esp. being on the right when facing the same direction as the observer.
left	<i>adj.</i> located nearer to this side of the body than the right.
red	<i>n.</i> the color of blood or a ruby.
blood	<i>n.</i> the red liquid that circulates in the heart, arteries and veins of animals.

Note the circularity in these definitions. The definition of *right* makes two direct references to itself, and the entry for *left* contains an implicit self-reference in the phrase *this side of the body*, which presumably means the *left* side. The entries for *red* and *blood* reference each other in their definitions. For humans, such entries are useful since the user of the dictionary has sufficient grasp of these other terms.

Yet despite their circularity and lack of formal representation, glosses can still be useful for computational modeling of senses. This is because a gloss is just a sentence, and from sentences we can compute sentence embeddings that tell us something about the meaning of the sense. Dictionaries often give example sentences along with glosses, and these can again be used to help build a sense representation.

The second way that thesauruses offer for defining a sense is—like the dictionary definitions—defining a sense through its relationship with other senses. For example, the above definitions make it clear that *right* and *left* are similar kinds of lemmas that stand in some kind of alternation, or opposition, to one another. Similarly, we can glean that *red* is a color and that *blood* is a *liquid*. **Sense relations** of this sort (**IS-A**, or **antonymy**) are explicitly listed in on-line databases like **WordNet**. Given a sufficiently large database of such relations, many applications are quite capable of performing sophisticated semantic tasks about word senses (even if they do not *really* know their right from their left).

19.1.2 How many senses do words have?

Dictionaries and thesauruses give discrete lists of senses. By contrast, embeddings (whether static or contextual) offer a continuous high-dimensional model of meaning that doesn't divide up into discrete senses.

Therefore creating a thesaurus depends on criteria for deciding when the differing uses of a word should be represented with discrete senses. We might consider two senses discrete if they have independent truth conditions, different syntactic behavior, and independent sense relations, or if they exhibit antagonistic meanings.

Consider the following uses of the verb *serve* from the WSJ corpus:

- (19.1) They rarely *serve* red meat, preferring to prepare seafood.
- (19.2) He *served* as U.S. ambassador to Norway in 1976 and 1977.
- (19.3) He might have *served* his time, come out and led an upstanding life.

zeugma

The *serve* of *serving red meat* and that of *serving time* clearly have different truth conditions and presuppositions; the *serve* of *serve as ambassador* has the distinct subcategorization structure *serve as NP*. These heuristics suggest that these are probably three distinct senses of *serve*. One practical technique for determining if two senses are distinct is to conjoin two uses of a word in a single sentence; this kind of conjunction of antagonistic readings is called **zeugma**. Consider the following examples:

- (19.4) Which of those flights serve breakfast?
- (19.5) Does Air France serve Philadelphia?
- (19.6) ?Does Air France serve breakfast and Philadelphia?

We use (?) to mark those examples that are semantically ill-formed. The oddness of the invented third example (a case of zeugma) indicates there is no sensible way to make a single sense of *serve* work for both breakfast and Philadelphia. We can use this as evidence that *serve* has two different senses in this case.

Dictionaries tend to use many fine-grained senses so as to capture subtle meaning differences, a reasonable approach given that the traditional role of dictionaries is aiding word learners. For computational purposes, we often don't need these fine distinctions, so we often group or cluster the senses; we have already done this for some of the examples in this chapter. Indeed, clustering examples into senses, or senses into broader-grained categories, is an important computational task that we'll discuss in Section 19.7.

19.2 Relations Between Senses

This section explores the relations between word senses, especially those that have received significant computational investigation like **synonymy**, **antonymy**, and **hyponymy**.

Synonymy

We introduced in Chapter 6 the idea that when two senses of two different words (**synonym**) are identical, or nearly identical, we say the two senses are **synonyms**. Synonyms include such pairs as

couch/sofa vomit/throw up filbert/hazelnut car/automobile

And we mentioned that in practice, the word *synonym* is commonly used to describe a relationship of approximate or rough synonymy. But furthermore, synonymy is actually a relationship between senses rather than words. Considering the words *big* and *large*. These may seem to be synonyms in the following sentences, since we could swap *big* and *large* in either sentence and retain the same meaning:

- (19.7) How big is that plane?
- (19.8) Would I be flying on a large or small plane?

But note the following sentence in which we cannot substitute *large* for *big*:

- (19.9) Miss Nelson, for instance, became a kind of big sister to Benjamin.
- (19.10) ?Miss Nelson, for instance, became a kind of large sister to Benjamin.

This is because the word *big* has a sense that means being older or grown up, while *large* lacks this sense. Thus, we say that some senses of *big* and *large* are (nearly) synonymous while other ones are not.

Antonymy

antonym Whereas synonyms are words with identical or similar meanings, **antonyms** are words with an opposite meaning, like:

*long/short big/little fast/slow cold/hot dark/light
rise/fall up/down in/out*

reversives Two senses can be antonyms if they define a binary opposition or are at opposite ends of some scale. This is the case for *long/short*, *fast/slow*, or *big/little*, which are at opposite ends of the *length* or *size* scale. Another group of antonyms, **reversives**, describe change or movement in opposite directions, such as *rise/fall* or *up/down*.

Antonyms thus differ completely with respect to one aspect of their meaning—their position on a scale or their direction—but are otherwise very similar, sharing almost all other aspects of meaning. Thus, automatically distinguishing synonyms from antonyms can be difficult.

Taxonomic Relations

hyponym Another way word senses can be related is taxonomically. A word (or sense) is a **hyponym** of another word or sense if the first is more specific, denoting a subclass of the other. For example, *car* is a hyponym of *vehicle*, *dog* is a hyponym of *animal*, and *mango* is a hyponym of *fruit*. Conversely, we say that *vehicle* is a **hypernym** of *car*, and *animal* is a hypernym of *dog*. It is unfortunate that the two words (hypernym

and hyponym) are very similar and hence easily confused; for this reason, the word **superordinate** is often used instead of **hypernym**.

Superordinate	vehicle	fruit	furniture	mammal
Subordinate	car	mango	chair	dog

We can define hypernymy more formally by saying that the class denoted by the superordinate extensionally includes the class denoted by the hyponym. Thus, the class of animals includes as members all dogs, and the class of moving actions includes all walking actions. Hypernymy can also be defined in terms of **entailment**. Under this definition, a sense *A* is a hyponym of a sense *B* if everything that is *A* is also *B*, and hence being an *A* entails being a *B*, or $\forall x A(x) \Rightarrow B(x)$. Hyponymy/hypernymy is usually a transitive relation; if *A* is a hyponym of *B* and *B* is a hyponym of *C*, then *A* is a hyponym of *C*. Another name for the hypernym/hyponym structure is the **IS-A** hierarchy, in which we say *A IS-A B*, or *B subsumes A*.

Hypernymy is useful for tasks like textual entailment or question answering; knowing that *leukemia* is a type of *cancer*, for example, would certainly be useful in answering questions about leukemia.

Meronymy

part-whole Another common relation is **meronymy**, the **part-whole** relation. A *leg* is part of a *chair*; a *wheel* is part of a *car*. We say that *wheel* is a **meronym** of *car*, and *car* is a **holonym** of *wheel*.

Structured Polysemy

The senses of a word can also be related semantically, in which case we call the relationship between them **structured polysemy**. Consider this sense *bank*:

(19.11) The bank is on the corner of Nassau and Witherspoon.

This sense, perhaps **bank**⁴, means something like “the building belonging to a financial institution”. These two kinds of senses (an organization and the building associated with an organization) occur together for many other words as well (*school*, *university*, *hospital*, etc.). Thus, there is a systematic relationship between senses that we might represent as

BUILDING ↔ ORGANIZATION

metonymy This particular subtype of polysemy relation is called **metonymy**. Metonymy is the use of one aspect of a concept or entity to refer to other aspects of the entity or to the entity itself. We are performing metonymy when we use the phrase *the White House* to refer to the administration whose office is in the White House. Other common examples of metonymy include the relation between the following pairings of senses:

AUTHOR (Jane Austen wrote <i>Emma</i>)	↔	WORKS OF AUTHOR (I really love Jane Austen)
FRUITTREE (Plums have beautiful blossoms)	↔	FRUIT (I ate a preserved plum yesterday)

19.3 WordNet: A Database of Lexical Relations

WordNet The most commonly used resource for sense relations in English and many other languages is the **WordNet** lexical database (Fellbaum, 1998). English WordNet consists of three separate databases, one each for nouns and verbs and a third for adjectives and adverbs; closed class words are not included. Each database contains a set of lemmas, each one annotated with a set of senses. The WordNet 3.0 release has 117,798 nouns, 11,529 verbs, 22,479 adjectives, and 4,481 adverbs. The average noun has 1.23 senses, and the average verb has 2.16 senses. WordNet can be accessed on the Web or downloaded locally. Figure 19.1 shows the lemma entry for the noun and adjective *bass*.

The noun “bass” has 8 senses in WordNet.

1. bass¹ - (the lowest part of the musical range)
2. bass², bass part¹ - (the lowest part in polyphonic music)
3. bass³, basso¹ - (an adult male singer with the lowest voice)
4. sea bass¹, bass⁴ - (the lean flesh of a saltwater fish of the family Serranidae)
5. freshwater bass¹, bass⁵ - (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
6. bass⁶, bass voice¹, basso² - (the lowest adult male singing voice)
7. bass⁷ - (the member with the lowest range of a family of musical instruments)
8. bass⁸ - (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Figure 19.1 A portion of the WordNet 3.0 entry for the noun *bass*.

gloss Note that there are eight senses for the noun and one for the adjective, each of which has a **gloss** (a dictionary-style definition), a list of synonyms for the sense, and sometimes also usage examples (shown for the adjective sense). WordNet doesn’t represent pronunciation, so doesn’t distinguish the pronunciation [b ae s] in **bass**⁴, **bass**⁵, and **bass**⁸ from the other senses pronounced [b ey s].

synset The set of near-synonyms for a WordNet sense is called a **synset** (for **synonym set**); synsets are an important primitive in WordNet. The entry for *bass* includes synsets like {*bass*¹, *deep*⁶}, or {*bass*⁶, *bass voice*¹, *basso*²}. We can think of a synset as representing a concept of the type we discussed in Chapter 16. Thus, instead of representing concepts in logical terms, WordNet represents them as lists of the word senses that can be used to express the concept. Here’s another synset example:

{*chump*¹, *fool*², *gull*¹, *mark*⁹, *patsy*¹, *fall guy*¹,
*sucker*¹, *soft touch*¹, *mug*²}

The gloss of this synset describes it as:

Gloss: a person who is gullible and easy to take advantage of.

Glosses are properties of a synset, so that each sense included in the synset has the same gloss and can express this concept. Because they share glosses, synsets like this one are the fundamental unit associated with WordNet entries, and hence it is synsets, not wordforms, lemmas, or individual senses, that participate in most of the lexical sense relations in WordNet.

WordNet also labels each synset with a lexicographic category drawn from a semantic field for example the 26 categories for nouns shown in Fig. 19.2, as well

supersense as 15 for verbs (plus 2 for adjectives and 1 for adverbs). These categories are often called **supersenses**, because they act as coarse semantic categories or groupings of senses which can be useful when word senses are too fine-grained (Ciaramita and Johnson 2003, Ciaramita and Altun 2006). Supersenses have also been defined for adjectives (Tsvetkov et al., 2014) and prepositions (Schneider et al., 2018).

Category	Example	Category	Example	Category	Example
ACT	<i>service</i>	GROUP	<i>place</i>	PLANT	<i>tree</i>
ANIMAL	<i>dog</i>	LOCATION	<i>area</i>	POSSESSION	<i>price</i>
ARTIFACT	<i>car</i>	MOTIVE	<i>reason</i>	PROCESS	<i>process</i>
ATTRIBUTE	<i>quality</i>	NATURAL EVENT	<i>experience</i>	QUANTITY	<i>amount</i>
BODY	<i>hair</i>	NATURAL OBJECT	<i>flower</i>	RELATION	<i>portion</i>
COGNITION	<i>way</i>	OTHER	<i>stuff</i>	SHAPE	<i>square</i>
COMMUNICATION	<i>review</i>	PERSON	<i>people</i>	STATE	<i>pain</i>
FEELING	<i>discomfort</i>	PHENOMENON	<i>result</i>	SUBSTANCE	<i>oil</i>
FOOD	<i>food</i>			TIME	<i>day</i>

Figure 19.2 Supersenses: 26 lexicographic categories for nouns in WordNet.

19.3.1 Sense Relations in WordNet

WordNet represents all the kinds of sense relations discussed in the previous section, as illustrated in Fig. 19.3 and Fig. 19.4.

Relation	Also Called	Definition	Example
Hypernym	Superordinate	From concepts to superordinates	<i>breakfast</i> ¹ → <i>meal</i> ¹
Hyponym	Subordinate	From concepts to subtypes	<i>meal</i> ¹ → <i>lunch</i> ¹
Instance Hypernym	Instance	From instances to their concepts	<i>Austen</i> ¹ → <i>author</i> ¹
Instance Hyponym	Has-Instance	From concepts to their instances	<i>composer</i> ¹ → <i>Bach</i> ¹
Part Meronym	Has-Part	From wholes to parts	<i>table</i> ² → <i>leg</i> ³
Part Holonym	Part-Of	From parts to wholes	<i>course</i> ⁷ → <i>meal</i> ¹
Antonym		Semantic opposition between lemmas	<i>leader</i> ¹ ⇔ <i>follower</i> ¹
Derivation		Lemmas w/same morphological root	<i>destruction</i> ¹ ⇔ <i>destroy</i> ¹

Figure 19.3 Some of the noun relations in WordNet.

Relation	Definition	Example
Hypernym	From events to superordinate events	<i>fly</i> ⁹ → <i>travel</i> ⁵
Troponym	From events to subordinate event	<i>walk</i> ¹ → <i>stroll</i> ¹
Entails	From verbs (events) to the verbs (events) they entail	<i>snore</i> ¹ → <i>sleep</i> ¹
Antonym	Semantic opposition between lemmas	<i>increase</i> ¹ ⇔ <i>decrease</i> ¹

Figure 19.4 Some verb relations in WordNet.

For example WordNet represents hyponymy (page 357) by relating each synset to its immediately more general and more specific synsets through direct hypernym and hyponym relations. These relations can be followed to produce longer chains of more general or more specific synsets. Figure 19.5 shows hypernym chains for **bass**³ and **bass**⁷; more general synsets are shown on successively indented lines.

WordNet has two kinds of taxonomic entities: classes and instances. An instance is an individual, a proper noun that is a unique entity. *San Francisco* is an instance of *city*, for example. But *city* is a class, a hyponym of *municipality* and eventually

```

bass3, basso (an adult male singer with the lowest voice)
=> singer, vocalist, vocalizer, vocaliser
=> musician, instrumentalist, player
=> performer, performing artist
=> entertainer
=> person, individual, someone...
=> organism, being
=> living thing, animate thing,
=> whole, unit
=> object, physical object
=> physical entity
=> entity

bass7 (member with the lowest range of a family of instruments)
=> musical instrument, instrument
=> device
=> instrumentality, instrumentation
=> artifact, artefact
=> whole, unit
=> object, physical object
=> physical entity
=> entity

```

Figure 19.5 Hyponymy chains for two separate senses of the lemma *bass*. Note that the chains are completely distinct, only converging at the very abstract level *whole, unit*.

of *location*. Fig. 19.6 shows a subgraph of WordNet demonstrating many of the relations.

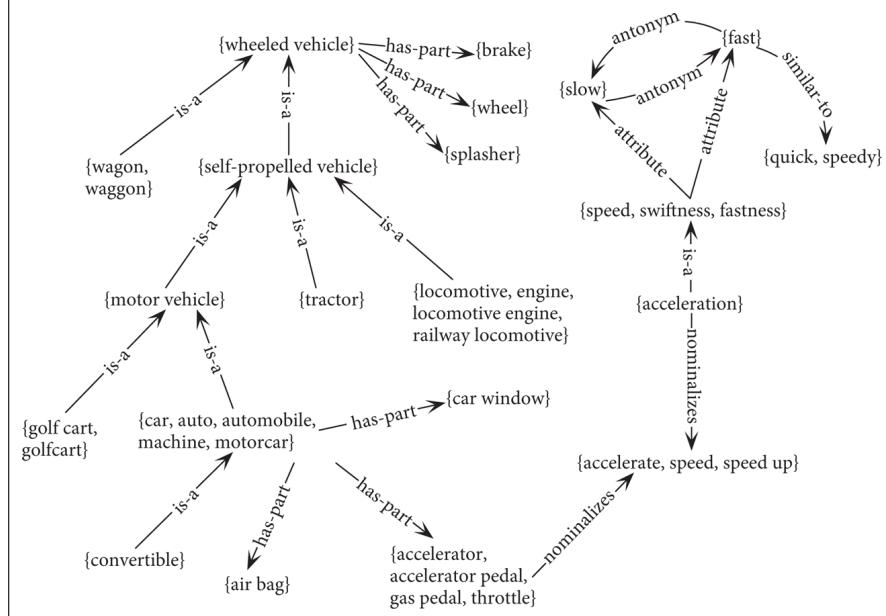


Figure 19.6 WordNet viewed as a graph. Figure from Navigli (2016).

19.4 Word Sense Disambiguation

**word sense
disambiguation
WSD**

The task of selecting the correct sense for a word is called **word sense disambiguation**, or **WSD**. WSD algorithms take as input a word in context and a fixed inventory of potential word senses and outputs the correct word sense in context.

19.4.1 WSD: The Task and Datasets

In this section we introduce the task setup for WSD, and then turn to algorithms. The inventory of sense tags depends on the task. For sense tagging in the context of translation from English to Spanish, the sense tag inventory for an English word might be the set of different Spanish translations. For automatic indexing of medical articles, the sense-tag inventory might be the set of MeSH (Medical Subject Headings) thesaurus entries. Or we can use the set of senses from a resource like WordNet, or supersenses if we want a coarser-grain set. Figure 19.4.1 shows some such examples for the word *bass*.

WordNet Sense	Spanish Translation	WordNet Supersense	Target Word in Context
bass ⁴	lubina	FOOD	... fish as Pacific salmon and striped bass and...
bass ⁷	bajo	ARTIFACT	... play bass because he doesn't have to solo...

Figure 19.7 Some possible sense tag inventories for *bass*.

lexical sample

In some situations, we just need to disambiguate a small number of words. In such **lexical sample** tasks, we have a small pre-selected set of target words and an inventory of senses for each word from some lexicon. Since the set of words and the set of senses are small, simple supervised classification approaches work very well.

all-words

More commonly, however, we have a harder problem in which we have to disambiguate all the words in some text. In this **all-words** task, the system is given an entire texts and a lexicon with an inventory of senses for each entry and we have to disambiguate every word in the text (or sometimes just every content word). The all-words task is similar to part-of-speech tagging, except with a much larger set of tags since each lemma has its own set. A consequence of this larger set of tags is data sparseness.

semantic concordance

Supervised **all-word** disambiguation tasks are generally trained from a **semantic concordance**, a corpus in which each open-class word in each sentence is labeled with its word sense from a specific dictionary or thesaurus, most often WordNet. The SemCor corpus is a subset of the Brown Corpus consisting of over 226,036 words that were manually tagged with WordNet senses (Miller et al. 1993, Landes et al. 1998). Other sense-tagged corpora have been built for the SENSEVAL and SemEval WSD tasks, such as the SENSEVAL-3 Task 1 English all-words test data with 2282 annotations (Snyder and Palmer, 2004) or the SemEval-13 Task 12 datasets. Large semantic concordances are also available in other languages including Dutch (Vossen et al., 2011) and German (Henrich et al., 2012).

Here's an example from the SemCor corpus showing the WordNet sense numbers of the tagged words; we've used the standard WSD notation in which a subscript marks the part of speech (Navigli, 2009):

(19.12) You will find⁹ that avocado_n¹ is_v¹ unlike_j¹ other_j¹ fruit_n¹ you have ever_r¹ tasted_v²

Given each noun, verb, adjective, or adverb word in the hand-labeled test set (say *fruit*), the SemCor-based WSD task is to choose the correct sense from the possible

senses in WordNet. For fruit this would mean choosing between the correct answer fruit¹_n (the ripened reproductive body of a seed plant), and the other two senses fruit²_n (yield; an amount of a product) and fruit³_n (the consequence of some effort or action). Fig. 19.8 sketches the task.

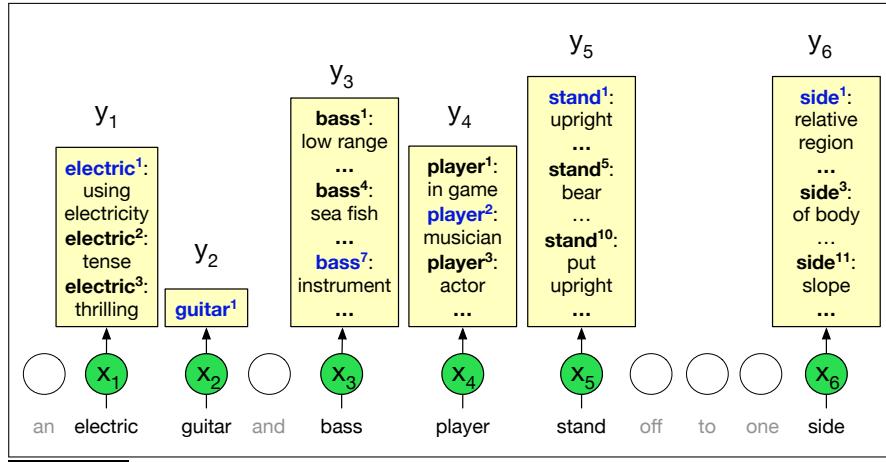


Figure 19.8 The all-words WSD task, mapping from input words (x) to WordNet senses (y). Only nouns, verbs, adjectives, and adverbs are mapped, and note that some words (like *guitar* in the example) only have one sense in WordNet. Figure inspired by [Chaplot and Salakhutdinov \(2018\)](#).

WSD systems are typically evaluated intrinsically, by computing F1 against hand-labeled sense tags in a held-out set, such as the SemCor corpus or SemEval corpora discussed above.

most frequent sense

A surprisingly strong baseline is simply to choose the **most frequent sense** for each word from the senses in a labeled corpus ([Gale et al., 1992a](#)). For WordNet, this corresponds to the first sense, since senses in WordNet are generally ordered from most frequent to least frequent based on their counts in the SemCor sense-tagged corpus. The most frequent sense baseline can be quite accurate, and is therefore often used as a default, to supply a word sense when a supervised algorithm has insufficient training data.

one sense per discourse

A second heuristic, called **one sense per discourse** is based on the work of [Gale et al. \(1992b\)](#), who noticed that a word appearing multiple times in a text or discourse often appears with the same sense. This heuristic seems to hold better for coarse-grained senses and particularly for cases of homonymy rather than polysemy, so isn't generally used as a baseline. Nonetheless various kinds of disambiguation tasks often include some such bias toward resolving an ambiguity the same way inside a discourse segment.

19.4.2 The WSD Algorithm: Contextual Embeddings

The best-performing WSD algorithm is a simple 1-nearest-neighbor algorithm using contextual word embeddings, due to [Melamud et al. \(2016\)](#) and [Peters et al. \(2018\)](#). At training time we pass each sentence in the SemCore labeled dataset through any contextual embedding (ELMo or BERT) resulting in a contextual embedding for each labeled token in SemCore. For each token c_i of each sense c of each word, we average the contextual representations to produce a contextual **sense embedding** \mathbf{v}_s

for c :

$$\mathbf{v}_s = \frac{1}{n} \sum_i \mathbf{c}_i \quad (19.13)$$

At test time we similarly compute a contextual embedding \mathbf{t} for the target word, and choose its nearest neighbor sense (the sense with the highest cosine with \mathbf{t}) from the training set. Fig. 19.9 illustrates the model.



Figure 19.9 The nearest-neighbor algorithm for WSD. In green are the contextual embeddings precomputed for each sense of each word; here we just show a few of the senses for *find*. A contextual embedding is computed for the target word *found*, and then the nearest neighbor sense (in this case find_n^9) would be chosen. Figure inspired by Loureiro and Jorge (2019).

What do we do for words we haven't seen in the sense-labeled training data? After all, the number of senses that appear in SemCor is only a small fraction of the words in WordNet. The simplest algorithm is to fall back to the Most Frequent Sense baseline, i.e. taking the first sense in WordNet. But that's not very satisfactory.

A more powerful approach, due to Loureiro and Jorge (2019), is to impute the missing sense embeddings, bottom-up, by using the WordNet taxonomy and supersenses. We get a sense embedding for any higher-level node in the WordNet taxonomy by averaging the embeddings of its children, thus computing the embedding for each synset as the average of its sense embeddings, the embedding for a hypernym as the average of its synset embeddings, and the lexicographic category (supersense) embedding as the average of the large set of synset embeddings with that category. More formally, for each missing sense in WordNet $\hat{s} \in W$, let the sense embeddings for the other members of its synset be $S_{\hat{s}}$, the hypernym-specific synset embeddings be $H_{\hat{s}}$, and the lexicographic (supersense-specific) synset embeddings be $L_{\hat{s}}$. We can then compute the sense embedding for \hat{s} as follows:

$$\text{if } |S_{\hat{s}}| > 0, \quad \mathbf{v}_{\hat{s}} = \frac{1}{|S_{\hat{s}}|} \sum \mathbf{v}_s, \forall \mathbf{v}_s \in S_{\hat{s}} \quad (19.14)$$

$$\text{else if } |H_{\hat{s}}| > 0, \quad \mathbf{v}_{\hat{s}} = \frac{1}{|H_{\hat{s}}|} \sum \mathbf{v}_{syn}, \forall \mathbf{v}_{syn} \in H_{\hat{s}} \quad (19.15)$$

$$\text{else if } |L_{\hat{s}}| > 0, \quad \mathbf{v}_{\hat{s}} = \frac{1}{|L_{\hat{s}}|} \sum \mathbf{v}_{syn}, \forall \mathbf{v}_{syn} \in L_{\hat{s}} \quad (19.16)$$

Since all of the supersenses have some labeled data in SemCor, the algorithm is guaranteed to have some representation for all possible senses by the time the al-

gorithm backs off to the most general (supersense) information, although of course with a very coarse model.

19.5 Alternate WSD algorithms and Tasks

19.5.1 Feature-Based WSD

Feature-based algorithms for WSD are extremely simple and function almost as well as contextual language model algorithms. The best-performing IMS algorithm ([Zhong and Ng, 2010](#)), augmented by embeddings ([Iacobacci et al. 2016](#), [Raganato et al. 2017b](#)), uses an SVM classifier to choose the sense for each input word with the following simple features of the surrounding words:

- part-of-speech tags (for a window of 3 words on each side, stopping at sentence boundaries)
- collocation** • **collocation** features of words or n-grams of lengths 1, 2, 3 at a particular location in a window of 3 word on each side (i.e., exactly one word to the right, or the two words starting 3 words to the left, and so on).
- weighted average of embeddings (of all words in a window of 10 words on each side, weighted exponentially by distance)

Consider the ambiguous word *bass* in the following WSJ sentence:

(19.17) An electric guitar and **bass** player stand off to one side,

If we used a small 2-word window, a standard feature vector might include parts-of-speech, unigram and bigram collocation features, and a weighted sum g of embeddings, that is:

$$[w_{i-2}, \text{POS}_{i-2}, w_{i-1}, \text{POS}_{i-1}, w_{i+1}, \text{POS}_{i+1}, w_{i+2}, \text{POS}_{i+2}, w_{i-2}^{i-1}, \\ w_{i+1}^{i+2}, g(E(w_{i-2}), E(w_{i-1}), E(w_{i+1}), E(w_{i+2}))] \quad (19.18)$$

would yield the following vector:

[guitar, NN, and, CC, player, NN, stand, VB, and guitar,
player stand, g(E(guitar), E(and), E(player), E(stand))]

19.5.2 The Lesk Algorithm as WSD Baseline

knowledge-based

Generating sense labeled corpora like SemCor is quite difficult and expensive. An alternative class of WSD algorithms, **knowledge-based** algorithms, rely solely on WordNet or other such resources and don't require labeled data. While supervised algorithms generally work better, knowledge-based methods can be used in languages or domains where thesauruses or dictionaries but not sense labeled corpora are available.

Lesk algorithm

The **Lesk algorithm** is the oldest and most powerful knowledge-based WSD method, and is a useful baseline. Lesk is really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood. Figure 19.10 shows the simplest version of the algorithm, often called the **Simplified Lesk** algorithm ([Kilgarriff and Rosenzweig, 2000](#)).

Simplified Lesk

As an example of the Lesk algorithm at work, consider disambiguating the word *bank* in the following context:

```

function SIMPLIFIED LESK(word, sentence) returns best sense of word
    best-sense  $\leftarrow$  most frequent sense for word
    max-overlap  $\leftarrow$  0
    context  $\leftarrow$  set of words in sentence
    for each sense in senses of word do
        signature  $\leftarrow$  set of words in the gloss and examples of sense
        overlap  $\leftarrow$  COMPUTEOVERLAP(signature, context)
        if overlap  $>$  max-overlap then
            max-overlap  $\leftarrow$  overlap
            best-sense  $\leftarrow$  sense
        end
    return(best-sense)

```

Figure 19.10 The Simplified Lesk algorithm. The COMPUTEOVERLAP function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the *context* in a more complex way.

(19.19) The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

given the following two WordNet senses:

bank ¹	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	“he cashed a check at the bank”, “that bank holds the mortgage on my home”
bank ²	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	“they pulled the canoe up on the bank”, “he sat on the bank of the river and watched the currents”

Sense **bank**¹ has two non-stopwords overlapping with the context in (19.19): *deposits* and *mortgage*, while sense **bank**² has zero words, so sense **bank**¹ is chosen.

There are many obvious extensions to Simplified Lesk, such as weighing the overlapping words by **IDF** (inverse document frequency) Chapter 6 to downweight frequent words like function words; best performing is to use word embedding cosine instead of word overlap to compute the similarity between the definition and the context (Basile et al., 2014). Modern neural extensions of Lesk use the definitions to compute sense embeddings that can be directly used instead of SemCor-training embeddings (Kumar et al. 2019, Luo et al. 2018a, Luo et al. 2018b).

19.5.3 Word-in-Context Evaluation

Word Sense Disambiguation is a much more fine-grained evaluation of word meaning than the context-free word similarity tasks we described in Chapter 6. Recall that tasks like LexSim-999 require systems to match human judgments on the context-free similarity between two words (how similar is *cup* to *mug*?). We can think of WSD as a kind of contextualized similarity task, since our goal is to be able to distinguish the meaning of a word like *bass* in one context (playing music) from another context (fishing).

word-in-context

Somewhere in between lies the **word-in-context** task. Here the system is given two sentences, each with the same target word but in a different sentential context. The system must decide whether the target words are used in the **same sense** in the

- wic two sentences or in a **different sense**. Fig. 19.11 shows sample pairs from the WiC dataset of [Pilehvar and Camacho-Collados \(2019\)](#).

-
- | | |
|---|---|
| F | There's a lot of trash on the bed of the river —
I keep a glass of water next to my bed when I sleep |
| F | Justify the margins — The end justifies the means |
| T | Air pollution — Open a window and let in some air |
| T | The expanded window will give us time to catch the thieves —
You have a two-hour window of clear weather to finish working on the lawn |
-

Figure 19.11 Positive (T) and negative (F) pairs from the WiC dataset ([Pilehvar and Camacho-Collados, 2019](#)).

The WiC sentences are mainly taken from the example usages for senses in WordNet. But WordNet senses are very fine-grained. For this reason tasks like word-in-context first cluster the word senses into coarser clusters, so that the two sentential contexts for the target word are marked as T if the two senses are in the same cluster. WiC clusters all pairs of senses if they are first degree connections in the WordNet semantic graph, including sister senses, or if they belong to the same supersense; we point to other sense clustering algorithms at the end of the chapter.

The baseline algorithm to solve the WIC task uses contextual embeddings like BERT with a simple thresholded cosine. We first compute the contextual embeddings for the target word in each of the two sentences, and then compute the cosine between them. If it's above a threshold tuned on a devset we respond true (the two senses are the same) else we respond false.

19.5.4 Wikipedia as a source of training data

Datasets other than SemCor have been used for all-words WSD. One important direction is to use Wikipedia as a source of sense-labeled data. When a concept is mentioned in a Wikipedia article, the article text may contain an explicit link to the concept's Wikipedia page, which is named by a unique identifier. This link can be used as a sense annotation. For example, the ambiguous word *bar* is linked to a different Wikipedia article depending on its meaning in context, including the page BAR (LAW), the page BAR (MUSIC), and so on, as in the following Wikipedia examples ([Mihalcea, 2007](#)).

In 1834, Sumner was admitted to the [[**bar (law)**]**|bar**] at the age of twenty-three, and entered private practice in Boston.

It is danced in 3/4 time (like most waltzes), with the couple turning approx. 180 degrees every [[**bar (music)**]**|bar**]].

Jenga is a popular beer in the [[**bar (establishment)**]**|bar**]]s of Thailand.

These sentences can then be added to the training data for a supervised system. In order to use Wikipedia in this way, however, it is necessary to map from Wikipedia concepts to whatever inventory of senses is relevant for the WSD application. Automatic algorithms that map from Wikipedia to WordNet, for example, involve finding the WordNet sense that has the greatest lexical overlap with the Wikipedia sense, by comparing the vector of words in the WordNet synset, gloss, and related senses with the vector of words in the Wikipedia page title, outgoing links, and page category ([Ponzetto andNavigli, 2010](#)). The resulting mapping has been used to create BabelNet, a large sense-annotated resource ([Navigli and Ponzetto, 2012](#)).

19.6 Using Thesauruses to Improve Embeddings

Thesauruses have also been used to improve both static and contextual word embeddings. For example, static word embeddings have a problem with **antonyms**. A word like *expensive* is often very similar in embedding cosine to its antonym like *cheap*. Antonymy information from thesauruses can help solve this problem; Fig. 19.12 shows nearest neighbors to some target words in GloVe, and the improvement after one such method.

Before counterfitting				After counterfitting		
east	west	north	south	eastward	eastern	easterly
expensive	pricey	cheaper	costly	costly	pricy	overpriced
British	American	Australian	Britain	Brits	London	BBC

Figure 19.12 The nearest neighbors in GloVe to *east*, *expensive*, and *British* include antonyms like *west*. The right side showing the improvement in GloVe nearest neighbors after the counterfitting method (Mrkšić et al., 2016).

There are two families of solutions. The first requires retraining: we modify the embedding training to incorporate thesaurus relations like synonymy, antonym, or supersenses. This can be done by modifying the static embedding loss function for word2vec (Yu and Dredze 2014, Nguyen et al. 2016) or by modifying contextual embedding training (Levine et al. 2019, Lauscher et al. 2019).

retrofitting

The second, for static embeddings, is more light-weight; after the embeddings have been trained we learn a second mapping based on a thesaurus that shifts the embeddings of words in such a way that synonyms (according to the thesaurus) are pushed closer and antonyms further apart. Such methods are called **retrofitting** (Faruqui et al. 2015, Lengerich et al. 2018) or **counterfitting** (Mrkšić et al., 2016).

19.7 Word Sense Induction

word sense induction

It is expensive and difficult to build large corpora in which each word is labeled for its word sense. For this reason, an unsupervised approach to sense disambiguation, often called **word sense induction** or **WSI**, is an important direction. In unsupervised approaches, we don't use human-defined word senses. Instead, the set of “senses” of each word is created automatically from the instances of each word in the training set.

Most algorithms for word sense induction follow the early work of Schütze (Schütze 1992b, Schütze 1998) in using some sort of clustering over word embeddings. In training, we use three steps:

1. For each token w_i of word w in a corpus, compute a context vector \mathbf{c} .
2. Use a **clustering algorithm** to **cluster** these word-token context vectors \mathbf{c} into a predefined number of groups or clusters. Each cluster defines a sense of w .
3. Compute the **vector centroid** of each cluster. Each vector centroid \mathbf{s}_j is a **sense vector** representing that sense of w .

Since this is an unsupervised algorithm, we don't have names for each of these “senses” of w ; we just refer to the j th sense of w .

To disambiguate a particular token t of w we again have three steps:

1. Compute a context vector \mathbf{c} for t .
2. Retrieve all sense vectors s_j for w .
3. Assign t to the sense represented by the sense vector s_j that is closest to t .

All we need is a clustering algorithm and a distance metric between vectors. Clustering is a well-studied problem with a wide number of standard algorithms that can be applied to inputs structured as vectors of numerical values (Duda and Hart, 1973). A frequently used technique in language applications is known as **agglomerative clustering**. In this technique, each of the N training instances is initially assigned to its own cluster. New clusters are then formed in a bottom-up fashion by the successive merging of the two clusters that are most similar. This process continues until either a specified number of clusters is reached, or some global goodness measure among the clusters is achieved. In cases in which the number of training instances makes this method too expensive, random sampling can be used on the original training set to achieve similar results.

agglomerative clustering

How can we evaluate unsupervised sense disambiguation approaches? As usual, the best way is to do extrinsic evaluation embedded in some end-to-end system; one example used in a SemEval bakeoff is to improve search result clustering and diversification (Navigli and Vannella, 2013). Intrinsic evaluation requires a way to map the automatically derived sense classes into a hand-labeled gold-standard set so that we can compare a hand-labeled test set with a set labeled by our unsupervised classifier. Various such metrics have been tested, for example in the SemEval tasks (Manandhar et al. 2010, Navigli and Vannella 2013, Jurgens and Klapaftis 2013), including cluster overlap metrics, or methods that map each sense cluster to a pre-defined sense by choosing the sense that (in some training set) has the most overlap with the cluster. However it is fair to say that no evaluation metric for this task has yet become standard.

19.8 Summary

This chapter has covered a wide range of issues concerning the meanings associated with lexical items. The following are among the highlights:

- A **word sense** is the locus of word meaning; definitions and meaning relations are defined at the level of the word sense rather than wordforms.
- Many words are **polysemous**, having many senses.
- Relations between senses include **synonymy**, **antonymy**, **meronymy**, and taxonomic relations **hyponymy** and **hypernymy**.
- **WordNet** is a large database of lexical relations for English, and WordNets exist for a variety of languages.
- **Word-sense disambiguation (WSD)** is the task of determining the correct sense of a word in context. Supervised approaches make use of a corpus of sentences in which individual words (**lexical sample task**) or all words (**all-words task**) are hand-labeled with senses from a resource like WordNet. SemCor is the largest corpus with WordNet-labeled senses.
- The standard supervised algorithm for WSD is nearest neighbors with contextual embeddings.
- Feature-based algorithms using parts of speech and embeddings of words in the context of the target word also work well.

- An important baseline for WSD is the **most frequent sense**, equivalent, in WordNet, to **take the first sense**.
- Another baseline is a **knowledge-based** WSD algorithm called the **Lesk algorithm** which chooses the sense whose dictionary definition shares the most words with the target word's neighborhood.
- **Word sense induction** is the task of learning word senses unsupervised.

Bibliographical and Historical Notes

Word sense disambiguation traces its roots to some of the earliest applications of digital computers. The insight that underlies modern algorithms for word sense disambiguation was first articulated by [Weaver \(1955\)](#) in the context of machine translation:

If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. [...] But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word. [...] The practical question is : “What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?”

Other notions first proposed in this early period include the use of a thesaurus for disambiguation ([Masterman, 1957](#)), supervised training of Bayesian models for disambiguation ([Madhu and Lytel, 1965](#)), and the use of clustering in word sense analysis ([Sparck Jones, 1986](#)).

An enormous amount of work on disambiguation was conducted within the context of early AI-oriented natural language processing systems. [Quillian \(1968\)](#) and [Quillian \(1969\)](#) proposed a graph-based approach to language understanding, in which the dictionary definition of words was represented by a network of word nodes connected by syntactic and semantic relations. He then proposed to do sense disambiguation by finding the shortest path between senses in the conceptual graph. [Simmons \(1973\)](#) is another influential early semantic network approach. Wilks proposed one of the earliest non-discrete models with his *Preference Semantics* ([Wilks 1975c](#), [Wilks 1975b](#), [Wilks 1975a](#)), and [Small and Rieger \(1982\)](#) and [Riesbeck \(1975\)](#) proposed understanding systems based on modeling rich procedural information for each word. Hirst's ABSITY system ([Hirst and Charniak 1982](#), [Hirst 1987](#), [Hirst 1988](#)), which used a technique called marker passing based on semantic networks, represents the most advanced system of this type. As with these largely symbolic approaches, early neural network (at the time called ‘connectionist’) approaches to word sense disambiguation relied on small lexicons with hand-coded representations ([Cottrell 1985](#), [Kawamoto 1988](#)).

The earliest implementation of a robust empirical approach to sense disambiguation is due to [Kelly and Stone \(1975\)](#), who directed a team that hand-crafted a set of disambiguation rules for 1790 ambiguous English words. [Lesk \(1986\)](#) was the first to use a machine-readable dictionary for word sense disambiguation. A collection of work concerning WordNet can be found in [Fellbaum \(1998\)](#). Early work using

dictionaries as lexical resources include Amsler's (1981) use of the Merriam Webster dictionary and Longman's *Dictionary of Contemporary English* (Boguraev and Briscoe, 1989).

Supervised approaches to disambiguation began with the use of decision trees by Black (1988). In addition to the IMS and contextual-embedding based methods for supervised WSD, recent supervised algorithms includes encoder-decoder models (Raganato et al., 2017a).

The need for large amounts of annotated text in supervised methods led early on to investigations into the use of bootstrapping methods (Hearst 1991, Yarowsky 1995). For example Diab and Resnik (2002) give a semi-supervised algorithm for sense disambiguation based on aligned parallel corpora in two languages. For example, the fact that the French word *catastrophe* might be translated as English *disaster* in one instance and *tragedy* in another instance can be used to disambiguate the senses of the two English words (i.e., to choose senses of *disaster* and *tragedy* that are similar).

coarse senses The earliest use of clustering in the study of word senses was by Sparck Jones (1986); Pedersen and Bruce (1997), Schütze (1997b), and Schütze (1998) applied distributional methods. Clustering word senses into **coarse senses** has also been used to address the problem of dictionary senses being too fine-grained (Section 19.5.3) (Dolan 1994, Chen and Chang 1998, Mihalcea and Moldovan 2001, Agirre and de Lacalle 2003, Palmer et al. 2004, Navigli 2006, Snow et al. 2007, Pilehvar et al. 2013). Corpora with clustered word senses for training supervised clustering algorithms include Palmer et al. (2006) and **OntoNotes** (Hovy et al., 2006).

OntoNotes Historical overviews of WSD include Agirre and Edmonds (2006) and Navigli (2009).

generative lexicon qualia structure See Pustejovsky (1995), Pustejovsky and Boguraev (1996), Martin (1986), and Copestake and Briscoe (1995), inter alia, for computational approaches to the representation of polysemy. Pustejovsky's theory of the **generative lexicon**, and in particular his theory of the **qualia structure** of words, is a way of accounting for the dynamic systematic polysemy of words in context.

Exercises

- 19.1 Collect a small corpus of example sentences of varying lengths from any newspaper or magazine. Using WordNet or any standard dictionary, determine how many senses there are for each of the open-class words in each sentence. How many distinct combinations of senses are there for each sentence? How does this number seem to vary with sentence length?
- 19.2 Using WordNet or a standard reference dictionary, tag each open-class word in your corpus with its correct tag. Was choosing the correct sense always a straightforward task? Report on any difficulties you encountered.
- 19.3 Using your favorite dictionary, simulate the original Lesk word overlap disambiguation algorithm described on page 366 on the phrase *Time flies like an arrow*. Assume that the words are to be disambiguated one at a time, from left to right, and that the results from earlier decisions are used later in the process.

- 19.4** Build an implementation of your solution to the previous exercise. Using WordNet, implement the original Lesk word overlap disambiguation algorithm described on page ?? on the phrase *Time flies like an arrow*.

Semantic Role Labeling

Sometime between the 7th and 4th centuries BCE, the Indian grammarian Pāṇini¹ wrote a famous treatise on Sanskrit grammar, the *Aṣṭādhyāyī* ('8 books'), a treatise that has been called "one of the greatest monuments of human intelligence" (Bloomfield, 1933, 11). The work describes the linguistics of the Sanskrit language in the form of 3959 sutras, each very efficiently (since it had to be memorized!) expressing part of a formal rule system that brilliantly prefigured modern mechanisms of formal language theory (Penn and Kiparsky, 2012). One set of rules, relevant to our discussion in this chapter, describes the **kārakas**, semantic relationships between a verb and noun arguments, roles like *agent*, *instrument*, or *destination*. Pāṇini's work was the earliest we know of that tried to understand the linguistic realization of events and their participants. This task of understanding how participants relate to events—being able to answer the question "Who did what to whom" (and perhaps also "when and where")—is a central question of natural language understanding.



Let's move forward 2.5 millennia to the present and consider the very mundane goal of understanding text about a purchase of stock by XYZ Corporation. This purchasing event and its participants can be described by a wide variety of surface forms. The event can be described by a verb (*sold*, *bought*) or a noun (*purchase*), and XYZ Corp can be the syntactic subject (of *bought*), the indirect object (of *sold*), or in a genitive or noun compound relation (with the noun *purchase*) despite having notionally the same role in all of them:

- XYZ corporation bought the stock.
- They sold the stock to XYZ corporation.
- The stock was bought by XYZ corporation.
- The purchase of the stock by XYZ corporation...
- The stock purchase by XYZ corporation...

In this chapter we introduce a level of representation that captures the commonality between these sentences: there was a purchase event, the participants were XYZ Corp and some stock, and XYZ Corp was the buyer. These shallow semantic representations, **semantic roles**, express the role that arguments of a predicate take in the event, codified in databases like PropBank and FrameNet. We'll introduce **semantic role labeling**, the task of assigning roles to spans in sentences, and **selectional restrictions**, the preferences that predicates express about their arguments, such as the fact that the theme of *eat* is generally something edible.

¹ Figure shows a birch bark manuscript from Kashmir of the Rupavatra, a grammatical textbook based on the Sanskrit grammar of Panini. Image from the Wellcome Collection.

20.1 Semantic Roles

Consider how in Chapter 16 we represented the meaning of arguments for sentences like these:

- (20.1) Sasha broke the window.
- (20.2) Pat opened the door.

A neo-Davidsonian event representation of these two sentences would be

$$\begin{aligned} \exists e, x, y \text{ } & \text{Breaking}(e) \wedge \text{Breaker}(e, \text{Sasha}) \\ & \wedge \text{BrokenThing}(e, y) \wedge \text{Window}(y) \\ \exists e, x, y \text{ } & \text{Opening}(e) \wedge \text{Opener}(e, \text{Pat}) \\ & \wedge \text{OpenedThing}(e, y) \wedge \text{Door}(y) \end{aligned}$$

deep roles In this representation, the roles of the subjects of the verbs *break* and *open* are *Breaker* and *Opener* respectively. These **deep roles** are specific to each event; *Breaking* events have *Breakers*, *Opening* events have *Openers*, and so on.

If we are going to be able to **answer questions**, perform inferences, or do any further kinds of natural language understanding of these events, we'll need to know a little more about the semantics of these arguments. *Breakers* and *Openers* have something in common. They are both volitional actors, often animate, and they have direct causal responsibility for their events.

thematic roles **agents** **Thematic roles** are a way to capture this semantic commonality between *Breakers* and *Eaters*. We say that the subjects of both these verbs are **agents**. Thus, **AGENT** is the thematic role that represents an abstract idea such as volitional causation. Similarly, the direct objects of both these verbs, the *BrokenThing* and *OpenedThing*, are both prototypically inanimate objects that are affected in some way by the action. **theme** The semantic role for these participants is **theme**.

Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
CONTENT	The proposition or content of a propositional event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

Figure 20.1 Some commonly used thematic roles with their definitions.

semantic roles Although thematic roles are one of the oldest linguistic models, as we saw above, their modern formulation is due to [Fillmore \(1968\)](#) and [Gruber \(1965\)](#). Although there is no universally agreed-upon set of roles, Figs. 20.1 and 20.2 list some thematic roles that have been used in various [computational papers](#), together with rough definitions and examples. Most thematic role sets have about a dozen roles, but we'll see sets with smaller numbers of roles with even more abstract meanings, and sets with very large numbers of roles that are specific to situations. We'll use the general term **semantic roles** for all sets of roles, whether small or large.

Thematic Role	Example
AGENT	<i>The waiter</i> spilled the soup.
EXPERIENCER	<i>John</i> has a headache.
FORCE	<i>The wind</i> blows debris from the mall into our yards.
THEME	Only after Benjamin Franklin broke <i>the ice</i> ...
RESULT	The city built a <i>regulation-size baseball diamond</i> ...
CONTENT	Mona asked “ <i>You met Mary Ann at a supermarket?</i> ”
INSTRUMENT	He poached catfish, stunning them with a <i>shocking device</i> ...
BENEFICIARY	Whenever Ann Callahan makes hotel reservations <i>for her boss</i> ...
SOURCE	I flew in <i>from Boston</i> .
GOAL	I drove <i>to Portland</i> .

Figure 20.2 Some prototypical examples of various thematic roles.

20.2 Diathesis Alternations

The main reason computational systems use semantic roles is to act as a shallow meaning representation that can let us make simple inferences that aren't possible from the pure surface string of words, or even from the parse tree. To extend the earlier examples, if a document says that *Company A acquired Company B*, we'd like to know that this answers the query *Was Company B acquired?* despite the fact that the two sentences have very different surface syntax. Similarly, this shallow semantics might act as a useful intermediate language in machine translation.

Semantic roles thus help generalize over different surface realizations of predicate arguments. For example, while the AGENT is often realized as the subject of the sentence, in other cases the THEME can be the subject. Consider these possible realizations of the thematic arguments of the verb *break*:

- (20.3) *John* *broke the window.*
 AGENT THEME
- (20.4) *John* *broke the window with a rock.*
 AGENT THEME INSTRUMENT
- (20.5) *The rock* *broke the window.*
 INSTRUMENT THEME
- (20.6) *The window* *broke.*
 THEME
- (20.7) *The window* *was broken by John.*
 THEME AGENT

These examples suggest that *break* has (at least) the possible arguments AGENT, THEME, and INSTRUMENT. The set of thematic role arguments taken by a verb is often called the **thematic grid**, θ -grid, or **case frame**. We can see that there are (among others) the following possibilities for the realization of these arguments of *break*:

- AGENT/Subject, THEME/Object
- AGENT/Subject, THEME/Object, INSTRUMENT/PP with
INSTRUMENT/Subject, THEME/Object
- THEME/Subject

thematic grid
case frame

It turns out that many verbs allow their thematic roles to be realized in various syntactic positions. For example, verbs like *give* can realize the THEME and GOAL arguments in two different ways:

- (20.8) a. *Doris gave the book to Cary.*

AGENT THEME GOAL

- b. *Doris gave Cary the book.*

AGENT GOAL THEME

These multiple argument structure realizations (the fact that *break* can take AGENT, INSTRUMENT, or THEME as subject, and *give* can realize its THEME and GOAL in either order) are called **verb alternations** or **diathesis alternations**. The alternation we showed above for *give*, the **dative alternation**, seems to occur with particular semantic classes of verbs, including “verbs of future having” (*advance, allocate, offer, owe*), “send verbs” (*forward, hand, mail*), “verbs of throwing” (*kick, pass, throw*), and so on. [Levin \(1993\)](#) lists for 3100 English verbs the semantic classes to which they belong (47 high-level classes, divided into 193 more specific classes) and the various alternations in which they participate. These lists of verb classes have been incorporated into the online resource VerbNet ([Kipper et al., 2000](#)), which links each verb to both WordNet and FrameNet entries.

20.3 Semantic Roles: Problems with Thematic Roles

Representing meaning at the thematic role level seems like it should be useful in dealing with complications like diathesis alternations. Yet it has proved quite difficult to come up with a standard set of roles, and equally difficult to produce a formal definition of roles like AGENT, THEME, or INSTRUMENT.

For example, researchers attempting to define role sets often find they need to fragment a role like AGENT or THEME into many specific roles. [Levin and Rappaport Hovav \(2005\)](#) summarize a number of such cases, such as the fact there seem to be at least two kinds of INSTRUMENTS, *intermediary* instruments that can appear as subjects and *enabling* instruments that cannot:

- (20.9) a. The cook opened the jar with the new gadget.
b. The new gadget opened the jar.

- (20.10) a. Shelly ate the sliced banana with a fork.
b. *The fork ate the sliced banana.

In addition to the fragmentation problem, there are cases in which we'd like to reason about and generalize across semantic roles, but the finite discrete lists of roles don't let us do this.

Finally, it has proved difficult to formally define the thematic roles. Consider the AGENT role; most cases of AGENTS are animate, volitional, sentient, causal, but any individual noun phrase might not exhibit all of these properties.

These problems have led to alternative **semantic role** models that use either many fewer or many more roles.

The first of these options is to define **generalized semantic roles** that abstract over the specific thematic roles. For example, PROTO-AGENT and PROTO-PATIENT are generalized roles that express roughly agent-like and roughly patient-like meanings. These roles are defined, not by necessary and sufficient conditions, but rather by a set of heuristic features that accompany more agent-like or more patient-like meanings. Thus, the more an argument displays agent-like properties (being volitionally involved in the event, causing an event or a change of state in another participant, being sentient or intentionally involved, moving) the greater the likelihood

verb
alternation
dative
alternation

semantic role

proto-agent
proto-patient

that the argument can be labeled a PROTO-AGENT. The more patient-like the properties (undergoing change of state, causally affected by another participant, stationary relative to other participants, etc.), the greater the likelihood that the argument can be labeled a PROTO-PATIENT.

The second direction is instead to define semantic roles that are specific to a particular verb or a particular group of semantically related verbs or nouns.

In the next two sections we describe two commonly used lexical resources that make use of these alternative versions of semantic roles. **PropBank** uses both proto-roles and verb-specific semantic roles. **FrameNet** uses semantic roles that are specific to a general semantic idea called a *frame*.

20.4 The Proposition Bank

PropBank

The **Proposition Bank**, generally referred to as **PropBank**, is a resource of sentences annotated with semantic roles. The English PropBank labels all the sentences in the Penn TreeBank; the Chinese PropBank labels sentences in the Penn Chinese TreeBank. Because of the difficulty of defining a universal set of thematic roles, the semantic roles in PropBank are defined with respect to an individual verb sense. Each sense of each verb thus has a specific set of roles, which are given only numbers rather than names: **Arg0**, **Arg1**, **Arg2**, and so on. In general, **Arg0** represents the PROTO-AGENT, and **Arg1**, the PROTO-PATIENT. The semantics of the other roles are less consistent, often being defined specifically for each verb. Nonetheless there are some generalization; the **Arg2** is often the benefactive, instrument, attribute, or end state, the **Arg3** the start point, benefactive, instrument, or attribute, and the **Arg4** the end point.

Here are some slightly simplified PropBank entries for one sense each of the verbs *agree* and *fall*. Such PropBank entries are called **frame files**; note that the definitions in the frame file for each role (“Other entity agreeing”, “Extent, amount fallen”) are informal glosses intended to be read by humans, rather than being formal definitions.

(20.11) **agree.01**

- Arg0: Agreer
- Arg1: Proposition
- Arg2: Other entity agreeing

- Ex1: [Arg0 The group] *agreed* [Arg1 it wouldn't make an offer].
- Ex2: [ArgM-TMP Usually] [Arg0 John] *agrees* [Arg2 with Mary]
[Arg1 on everything].

(20.12) **fall.01**

- Arg1: Logical subject, patient, thing falling
- Arg2: Extent, amount fallen
- Arg3: start point
- Arg4: end point, end state of arg1
- Ex1: [Arg1 Sales] *fell* [Arg4 to \$25 million] [Arg3 from \$27 million].
- Ex2: [Arg1 The average junk bond] *fell* [Arg2 by 4.2%].

Note that there is no Arg0 role for *fall*, because the normal subject of *fall* is a PROTO-PATIENT.

The PropBank semantic roles can be useful in recovering shallow semantic information about verbal arguments. Consider the verb *increase*:

- (20.13) **increase.01** “go up incrementally”

Arg0: causer of increase
 Arg1: thing increasing
 Arg2: amount increased by, EXT, or MNR
 Arg3: start point
 Arg4: end point

A PropBank semantic role labeling would allow us to infer the commonality in the event structures of the following three examples, that is, that in each case *Big Fruit Co.* is the AGENT and *the price of bananas* is the THEME, despite the differing surface forms.

- (20.14) [Arg0 Big Fruit Co.] increased [Arg1 the price of bananas].

- (20.15) [Arg1 The price of bananas] was increased again [Arg0 by Big Fruit Co.]

- (20.16) [Arg1 The price of bananas] increased [Arg2 5%].

PropBank also has a number of non-numbered arguments called **ArgMs**, (ArgM-TMP, ArgM-LOC, etc.) which represent modification or adjunct meanings. These are relatively stable across predicates, so aren’t listed with each frame file. Data labeled with these modifiers can be helpful in training systems to detect temporal, location, or directional modification across predicates. Some of the ArgM’s include:

TMP	when?	yesterday evening, now
LOC	where?	at the museum, in San Francisco
DIR	where to/from?	down, to Bangkok
MNR	how?	clearly, with much enthusiasm
PRP/CAU	why?	because ... , in response to the ruling
REC		themselves, each other
ADV	miscellaneous	
PRD	secondary predication	...ate the meat raw

NomBank

While PropBank focuses on verbs, a related project, **NomBank** (Meyers et al., 2004) adds annotations to noun predicates. For example the noun *agreement* in *Apple’s agreement with IBM* would be labeled with Apple as the Arg0 and IBM as the Arg2. This allows semantic role labelers to assign labels to arguments of both verbal and nominal predicates.

20.5 FrameNet

While making inferences about the semantic commonalities across different sentences with *increase* is useful, it would be even more useful if we could make such inferences in many more situations, across different verbs, and also between verbs and nouns. For example, we’d like to extract the similarity among these three sentences:

- (20.17) [Arg1 The price of bananas] increased [Arg2 5%].

- (20.18) [Arg1 The price of bananas] rose [Arg2 5%].

- (20.19) There has been a [Arg2 5%] rise [Arg1 in the price of bananas].

Note that the second example uses the different verb *rise*, and the third example uses the noun rather than the verb *rise*. We’d like a system to recognize that *the*

FrameNet

price of bananas is what went up, and that 5% is the amount it went up, no matter whether the 5% appears as the object of the verb *increased* or as a nominal modifier of the noun *rise*.

The **FrameNet** project is another semantic-role-labeling project that attempts to address just these kinds of problems (Baker et al. 1998, Fillmore et al. 2003, Fillmore and Baker 2009, Ruppenhofer et al. 2016). Whereas roles in the PropBank project are specific to an individual verb, roles in the FrameNet project are specific to a **frame**.

What is a frame? Consider the following set of words:

reservation, flight, travel, buy, price, cost, fare, rates, meal, plane

There are many individual lexical relations of hyponymy, synonymy, and so on between many of the words in this list. The resulting set of relations does not, however, add up to a complete account of how these words are related. They are clearly all defined with respect to a coherent chunk of common-sense background information concerning air travel.

We call the holistic background knowledge that unites these words a **frame** (Fillmore, 1985). The idea that groups of words are defined with respect to some background information is widespread in artificial intelligence and cognitive science, where besides **frame** we see related works like a **model** (Johnson-Laird, 1983), or even **script** (Schank and Abelson, 1977).

A frame in FrameNet is a background knowledge structure that defines a set of frame-specific semantic roles, called **frame elements**, and includes a set of predicates that use these roles. Each word evokes a frame and profiles some aspect of the frame and its elements. The FrameNet dataset includes a set of frames and frame elements, the lexical units associated with each frame, and a set of labeled example sentences. For example, the **change_position_on_a_scale** frame is defined as follows:

This frame consists of words that indicate the change of an Item's position on a scale (the Attribute) from a starting point (Initial_value) to an end point (Final_value).

Some of the semantic roles (frame elements) in the frame are defined as in Fig. 20.3. Note that these are separated into **core roles**, which are frame specific, and **non-core roles**, which are more like the Arg-M arguments in PropBank, expressing more general properties of time, location, and so on.

Here are some example sentences:

- (20.20) [ITEM Oil] *rose* [ATTRIBUTE in price] [DIFFERENCE by 2%].
- (20.21) [ITEM It] has *increased* [FINAL_STATE to having them 1 day a month].
- (20.22) [ITEM Microsoft shares] *fell* [FINAL_VALUE to 7 5/8].
- (20.23) [ITEM Colon cancer incidence] *fell* [DIFFERENCE by 50%] [GROUP among men].
- (20.24) a steady *increase* [INITIAL_VALUE from 9.5] [FINAL_VALUE to 14.3] [ITEM in dividends]
- (20.25) a [DIFFERENCE 5%] [ITEM dividend] *increase*...

Note from these example sentences that the frame includes target words like *rise*, *fall*, and *increase*. In fact, the complete frame consists of the following words:

Core Roles	
ATTRIBUTE	The ATTRIBUTE is a scalar property that the ITEM possesses.
DIFFERENCE	The distance by which an ITEM changes its position on the scale.
FINAL_STATE	A description that presents the ITEM's state after the change in the ATTRIBUTE's value as an independent predication.
FINAL_VALUE	The position on the scale where the ITEM ends up.
INITIAL_STATE	A description that presents the ITEM's state before the change in the ATTRIBUTE's value as an independent predication.
INITIAL_VALUE	The initial position on the scale from which the ITEM moves away.
ITEM	The entity that has a position on the scale.
VALUE_RANGE	A portion of the scale, typically identified by its end points, along which the values of the ATTRIBUTE fluctuate.
Some Non-Core Roles	
DURATION	The length of time over which the change takes place.
SPEED	The rate of change of the VALUE.
GROUP	The GROUP in which an ITEM changes the value of an ATTRIBUTE in a specified way.

Figure 20.3 The frame elements in the `change_position_on_a_scale` frame from the FrameNet Labelers Guide (Ruppenhofer et al., 2016).

VERBS:	dwindle	move	soar	escalation	shift
advance	edge	mushroom	swell	explosion	tumble
climb	explode	plummet	swing	fall	
decline	fall	reach	triple	fluctuation	ADVERBS:
decrease	fluctuate	rise	tumble	gain	increasingly
diminish	gain	rocket		growth	
dip	grow	shift		NOUNS:	hike
double	increase	skyrocket	decline	increase	
drop	jump	slide	decrease	rise	

FrameNet also codes relationships between frames, allowing frames to inherit from each other, or representing relations between frames like causation (and generalizations among frame elements in different frames can be represented by inheritance as well). Thus, there is a `Cause_change_of_position_on_a_scale` frame that is linked to the `Change_of_position_on_a_scale` frame by the `cause` relation, but that adds an `AGENT` role and is used for causative examples such as the following:

(20.26) [AGENT They] *raised* [ITEM the price of their soda] [DIFFERENCE by 2%].

Together, these two frames would allow an understanding system to extract the common event semantics of all the verbal and nominal causative and non-causative usages.

FrameNets have also been developed for many other languages including Spanish, German, Japanese, Portuguese, Italian, and Chinese.

20.6 Semantic Role Labeling

semantic role labeling

Semantic role labeling (sometimes shortened as **SRL**) is the task of automatically finding the **semantic roles** of each argument of each predicate in a sentence. Current approaches to semantic role labeling are based on supervised machine learning, often using the FrameNet and PropBank resources to specify what counts as a predicate, define the set of roles used in the task, and provide training and test sets.

Recall that the difference between these two models of semantic roles is that FrameNet (20.27) employs many frame-specific frame elements as roles, while PropBank (20.28) uses a smaller number of numbered argument labels that can be interpreted as verb-specific labels, along with the more general ARG-M labels. Some examples:

- (20.27) [You] can't [blame] [the program] [for being unable to identify it]
COGNIZER TARGET EVALUER REASON
- (20.28) [The San Francisco Examiner] issued [a special edition] [yesterday]
ARGO TARGET ARG1 ARG-M-TMP

20.6.1 A Feature-based Algorithm for Semantic Role Labeling

A simplified feature-based semantic role labeling algorithm is sketched in Fig. 20.4. Feature-based algorithms—from the very earliest systems like (Simmons, 1973)—begin by parsing, using broad-coverage parsers to assign a parse to the input string. Figure 20.5 shows a parse of (20.28) above. The parse is then traversed to find all words that are predicates.

For each of these predicates, the algorithm examines each node in the parse tree and uses supervised classification to decide the semantic role (if any) it plays for this predicate. Given a labeled training set such as PropBank or FrameNet, a feature vector is extracted for each node, using feature templates described in the next subsection. A 1-of-N classifier is then trained to predict a semantic role for each constituent given these features, where N is the number of potential semantic roles plus an extra NONE role for non-role constituents. Any standard classification algorithms can be used. Finally, for each test sentence to be labeled, the classifier is run on each relevant constituent.

```

function SEMANTICROLELABEL(words) returns labeled tree
    parse  $\leftarrow$  PARSE(words)
    for each predicate in parse do
        for each node in parse do
            featurevector  $\leftarrow$  EXTRACTFEATURES(node, predicate, parse)
            CLASSIFYNODE(node, featurevector, parse)

```

Figure 20.4 A generic semantic-role-labeling algorithm. CLASSIFYNODE is a 1-of-*N* classifier that assigns a semantic role (or NONE for non-role constituents), trained on labeled data such as FrameNet or PropBank.

Instead of training a single-stage classifier as in Fig. 20.5, the node-level classification task can be broken down into multiple steps:

1. **Pruning:** Since only a small number of the constituents in a sentence are arguments of any given predicate, many systems use simple heuristics to prune unlikely constituents.
2. **Identification:** a binary classification of each node as an argument to be labeled or a NONE.
3. **Classification:** a 1-of-*N* classification of all the constituents that were labeled as arguments by the previous stage

The separation of identification and classification may lead to better use of features (different features may be useful for the two tasks) or to computational efficiency.

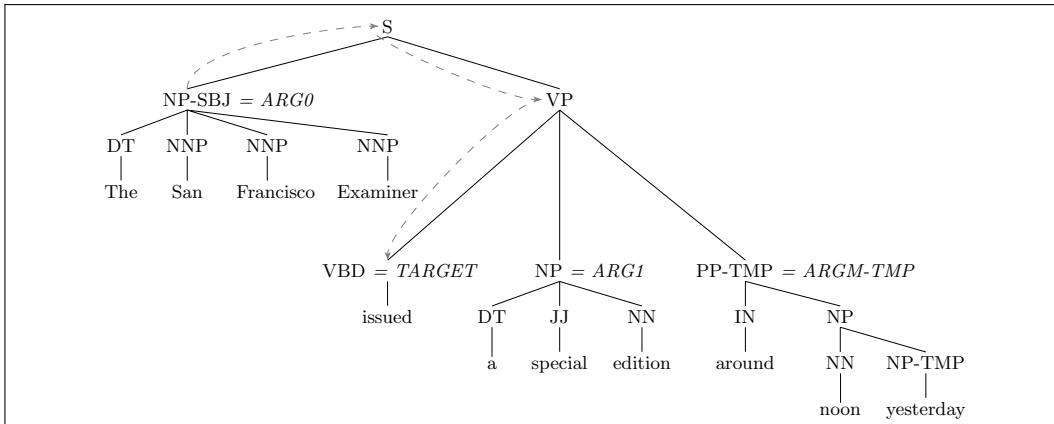


Figure 20.5 Parse tree for a PropBank sentence, showing the PropBank argument labels. The dotted line shows the **path** feature $NP \uparrow S \downarrow VP \downarrow VBD$ for $ARG0$, the NP-SBJ constituent *The San Francisco Examiner*.

Global Optimization

The classification algorithm of Fig. 20.5 classifies each argument separately (“locally”), making the simplifying assumption that each argument of a predicate can be labeled independently. This assumption is false; there are interactions between arguments that require a more ‘global’ assignment of labels to constituents. For example, constituents in FrameNet and PropBank are required to be non-overlapping. More significantly, the semantic roles of constituents are not independent. For example PropBank does not allow multiple identical arguments; two constituents of the same verb cannot both be labeled $ARG0$.

Role labeling systems thus often add a fourth step to deal with global consistency across the labels in a sentence. For example, the local classifiers can return a list of possible labels associated with probabilities for each constituent, and a second-pass Viterbi decoding or re-ranking approach can be used to choose the best consensus label. Integer linear programming (ILP) is another common way to choose a solution that conforms best to multiple constraints.

Features for Semantic Role Labeling

Most systems use some generalization of the core set of features introduced by Gildea and Jurafsky (2000). Common basic features templates (demonstrated on the *NP-SBJ* constituent *The San Francisco Examiner* in Fig. 20.5) include:

- The governing **predicate**, in this case the verb *issued*. The predicate is a crucial feature since labels are defined only with respect to a particular predicate.
- The **phrase type** of the constituent, in this case, *NP* (or *NP-SBJ*). Some semantic roles tend to appear as *NPs*, others as *S* or *PP*, and so on.
- The **headword** of the constituent, *Examiner*. The headword of a constituent can be computed with standard head rules, such as those given in Chapter 12 in Fig. 12.12. Certain headwords (e.g., pronouns) place strong constraints on the possible semantic roles they are likely to fill.
- The **headword part of speech** of the constituent, *NNP*.
- The **path** in the parse tree from the constituent to the predicate. This path is marked by the dotted line in Fig. 20.5. Following Gildea and Jurafsky (2000), we can use a simple linear representation of the path, $NP \uparrow S \downarrow VP \downarrow VBD$. \uparrow and \downarrow represent upward and downward movement in the tree, respectively. The

path is very useful as a compact representation of many kinds of grammatical function relationships between the constituent and the predicate.

- The **voice** of the clause in which the constituent appears, in this case, **active** (as contrasted with **passive**). Passive sentences tend to have strongly different linkings of semantic roles to surface form than do active ones.
- The binary **linear position** of the constituent with respect to the predicate, either **before** or **after**.
- The **subcategorization** of the predicate, the set of expected arguments that appear in the verb phrase. We can extract this information by using the phrase-structure rule that expands the immediate parent of the predicate; $\text{VP} \rightarrow \text{VBD} \text{ NP PP}$ for the predicate in Fig. 20.5.
- The named entity type of the constituent.
- The first words and the last word of the constituent.

The following feature vector thus represents the first NP in our example (recall that most observations will have the value NONE rather than, for example, ARG0, since most constituents in the parse tree will not bear a semantic role):

$\text{ARG0: [issued, NP, Examiner, NNP, } \text{NP} \uparrow \text{S} \downarrow \text{VP} \downarrow \text{VBD, active, before, VP} \rightarrow \text{NP PP, }$
 $\text{ORG, The, Examiner]}$

Other features are often used in addition, such as sets of n-grams inside the constituent, or more complex versions of the path features (the upward or downward halves, or whether particular nodes occur in the path).

It's also possible to use dependency parses instead of constituency parses as the basis of features, for example using dependency parse paths instead of constituency paths.

20.6.2 A Neural Algorithm for Semantic Role Labeling

The standard neural algorithm for semantic role labeling is based on the bi-LSTM IOB tagger introduced in Chapter 9, which we've seen applied to part-of-speech tagging and named entity tagging, among other tasks. Recall that with IOB tagging, we have a begin and end tag for each possible role (B-ARG0, I-ARG0; B-ARG1, I-ARG1, and so on), plus an outside tag O.

As with all the taggers, the goal is to compute the highest probability tag sequence \hat{y} , given the input sequence of words w :

$$\hat{y} = \underset{y \in T}{\operatorname{argmax}} P(\mathbf{y} | \mathbf{w})$$

In algorithms like He et al. (2017), each input word is mapped to pre-trained embeddings, and also associated with an embedding for a flag (0/1) variable indicating whether that input word is the predicate. These concatenated embeddings are passed through multiple layers of bi-directional LSTM. State-of-the-art algorithms tend to be deeper than for POS or NER tagging, using 3 to 4 layers (6 to 8 total LSTMs). Highway layers can be used to connect these layers as well.

Output from the last bi-LSTM can then be turned into an IOB sequence as for POS or NER tagging. Tags can be locally optimized by taking the bi-LSTM output, passing it through a single layer into a softmax for each word that creates a probability distribution over all SRL tags and the most likely tag for word x_i is chosen as t_i , computing for each word essentially:

$$\hat{t}_i = \underset{t \in \text{tags}}{\operatorname{argmax}} P(t | w_i)$$



Figure 20.6 A bi-LSTM approach to semantic role labeling. Most actual networks are much deeper than shown in this figure; 3 to 4 bi-LSTM layers (6 to 8 total LSTMs) are common. The input is a concatenation of an embedding for the input word and an embedding of a binary variable which is 1 for the predicate to 0 for all other words. After [He et al. \(2017\)](#).

However, just as feature-based SRL tagging, this local approach to decoding doesn't exploit the global constraints between tags; a tag I-ARG0, for example, must follow another I-ARG0 or B-ARG0.

As we saw for POS and NER tagging, there are many ways to take advantage of these global constraints. A CRF layer can be used instead of a softmax layer on top of the bi-LSTM output, and the Viterbi decoding algorithm can be used to decode from the CRF.

An even simpler Viterbi decoding algorithm that may perform equally well and doesn't require adding CRF complexity to the training process is to start with the simple softmax. The softmax output (the entire probability distribution over tags) for each word is then treated as a lattice and we can do Viterbi decoding through the lattice. The hard IOB constraints can act as the transition probabilities in the Viterbi decoding (thus the transition from state I-ARG0 to I-ARG1 would have probability 0). Alternatively, the training data can be used to learn bigram or trigram tag transition probabilities as if doing HMM decoding. Fig. 20.6 shows a sketch of the algorithm.

20.6.3 Evaluation of Semantic Role Labeling

The standard evaluation for semantic role labeling is to require that each argument label must be assigned to the exactly correct word sequence or parse constituent, and then compute precision, recall, and *F*-measure. Identification and classification can also be evaluated separately. Two common datasets used for evaluation are CoNLL-2005 ([Carreras and Márquez, 2005](#)) and CoNLL-2012 ([Pradhan et al., 2013](#)).

20.7 Selectional Restrictions

selectional restriction

We turn in this section to another way to represent facts about the relationship between predicates and arguments. A **selectional restriction** is a semantic type constraint that a verb imposes on the kind of concepts that are allowed to fill its argument roles. Consider the two meanings associated with the following example:

(20.29) I want to eat someplace nearby.

There are two possible parses and semantic interpretations for this sentence. In the sensible interpretation, *eat* is intransitive and the phrase *someplace nearby* is an adjunct that gives the location of the eating event. In the nonsensical *speaker-as-Godzilla* interpretation, *eat* is transitive and the phrase *someplace nearby* is the direct object and the THEME of the eating, like the NP *Malaysian food* in the following sentences:

(20.30) I want to eat Malaysian food.

How do we know that *someplace nearby* isn't the direct object in this sentence? One useful cue is the semantic fact that the THEME of EATING events tends to be something that is *edible*. This restriction placed by the verb *eat* on the filler of its THEME argument is a selectional restriction.

Selectional restrictions are associated with senses, not entire lexemes. We can see this in the following examples of the lexeme *serve*:

(20.31) The restaurant serves green-lipped mussels.

(20.32) Which airlines serve Denver?

Example (20.31) illustrates the offering-food sense of *serve*, which ordinarily restricts its THEME to be some kind of food Example (20.32) illustrates the *provides a commercial service to* sense of *serve*, which constrains its THEME to be some type of appropriate location.

Selectional restrictions vary widely in their specificity. The verb *imagine*, for example, imposes strict requirements on its AGENT role (restricting it to humans and other animate entities) but places very few semantic requirements on its THEME role. A verb like *diagonalize*, on the other hand, places a very specific constraint on the filler of its THEME role: it has to be a matrix, while the arguments of the adjectives *odorless* are restricted to concepts that could possess an odor:

(20.33) In rehearsal, I often ask the musicians to *imagine* a tennis game.

(20.34) Radon is an *odorless* gas that can't be detected by human senses.

(20.35) To *diagonalize* a matrix is to find its eigenvalues.

These examples illustrate that the set of concepts we need to represent selectional restrictions (being a matrix, being able to possess an odor, etc) is quite open ended. This distinguishes selectional restrictions from other features for representing lexical knowledge, like parts-of-speech, which are quite limited in number.

20.7.1 Representing Selectional Restrictions

One way to capture the semantics of selectional restrictions is to use and extend the event representation of Chapter 16. Recall that the neo-Davidsonian representation of an event consists of a single variable that stands for the event, a predicate denoting the kind of event, and variables and relations for the event roles. Ignoring the issue of the λ -structures and using thematic roles rather than deep event roles, the semantic contribution of a verb like *eat* might look like the following:

$$\exists e, x, y \text{ } Eating(e) \wedge \text{Agent}(e, x) \wedge \text{Theme}(e, y)$$

With this representation, all we know about *y*, the filler of the THEME role, is that it is associated with an *Eating* event through the *Theme* relation. To stipulate the

```

Sense 1
hamburger, beefburger --
(a fried cake of minced beef served on a bun)
=> sandwich
=> snack food
=> dish
=> nutrient, nourishment, nutrition...
=> food, nutrient
=> substance
=> matter
=> physical entity
=> entity

```

Figure 20.7 Evidence from WordNet that hamburgers are edible.

selectional restriction that y must be something edible, we simply add a new term to that effect:

$$\exists e, x, y \text{ } Eating(e) \wedge \text{Agent}(e, x) \wedge \text{Theme}(e, y) \wedge \text{EdibleThing}(y)$$

When a phrase like *ate a hamburger* is encountered, a semantic analyzer can form the following kind of representation:

$$\exists e, x, y \text{ } Eating(e) \wedge \text{Eater}(e, x) \wedge \text{Theme}(e, y) \wedge \text{EdibleThing}(y) \wedge \text{Hamburger}(y)$$

This representation is perfectly reasonable since the membership of y in the category *Hamburger* is consistent with its membership in the category *EdibleThing*, assuming a reasonable set of facts in the knowledge base. Correspondingly, the representation for a phrase such as *ate a takeoff* would be ill-formed because membership in an event-like category such as *Takeoff* would be inconsistent with membership in the category *EdibleThing*.

While this approach adequately captures the semantics of selectional restrictions, there are two problems with its direct use. First, using FOL to perform the simple task of enforcing selectional restrictions is overkill. Other, far simpler, formalisms can do the job with far less computational cost. The second problem is that this approach presupposes a large, logical knowledge base of facts about the concepts that make up selectional restrictions. Unfortunately, although such common-sense knowledge bases are being developed, none currently have the kind of coverage necessary to the task.

A more practical approach is to state selectional restrictions in terms of WordNet synsets rather than as logical concepts. Each predicate simply specifies a WordNet synset as the selectional restriction on each of its arguments. A meaning representation is well-formed if the role filler word is a hyponym (subordinate) of this synset.

For our *ate a hamburger* example, for instance, we could set the selectional restriction on the THEME role of the verb *eat* to the synset **{food, nutrient}**, glossed as *any substance that can be metabolized by an animal to give energy and build tissue*. Luckily, the chain of hypernyms for *hamburger* shown in Fig. 20.7 reveals that hamburgers are indeed food. Again, the filler of a role need not match the restriction synset exactly; it just needs to have the synset as one of its superordinates.

We can apply this approach to the THEME roles of the verbs *imagine*, *lift*, and *diagonalize*, discussed earlier. Let us restrict *imagine*'s THEME to the synset **{entity}**, *lift*'s THEME to **{physical entity}**, and *diagonalize* to **{matrix}**. This arrangement

correctly permits *imagine a hamburger* and *lift a hamburger*, while also correctly ruling out *diagonalize a hamburger*.

20.7.2 Selectional Preferences

In the earliest implementations, selectional restrictions were considered strict constraints on the kind of arguments a predicate could take (Katz and Fodor 1963, Hirst 1987). For example, the verb *eat* might require that its THEME argument be [+FOOD]. Early word sense disambiguation systems used this idea to rule out senses that violated the selectional restrictions of their governing predicates.

Very quickly, however, it became clear that these selectional restrictions were better represented as preferences rather than strict constraints (Wilks 1975c, Wilks 1975b). For example, selectional restriction violations (like inedible arguments of *eat*) often occur in well-formed sentences, for example because they are negated (20.36), or because selectional restrictions are overstated (20.37):

(20.36) But it fell apart in 1931, perhaps because people realized you can't **eat** gold for lunch if you're hungry.

(20.37) In his two championship trials, Mr. Kulkarni **ate** glass on an empty stomach, accompanied only by water and tea.

Modern systems for selectional preferences therefore specify the relation between a predicate and its possible arguments with soft constraints of some kind.

Selectional Association

selectional preference strength

relative entropy
KL divergence

One of the most influential has been the **selectional association** model of Resnik (1993). Resnik defines the idea of **selectional preference strength** as the general amount of information that a predicate tells us about the semantic class of its arguments. For example, the verb *eat* tells us a lot about the semantic class of its direct objects, since they tend to be edible. The verb *be*, by contrast, tells us less about its direct objects. The selectional preference strength can be defined by the difference in information between two distributions: the distribution of expected semantic classes $P(c)$ (how likely is it that a direct object will fall into class c) and the distribution of expected semantic classes for the particular verb $P(c|v)$ (how likely is it that the direct object of the specific verb v will fall into semantic class c). The greater the difference between these distributions, the more information the verb is giving us about possible objects. The difference between these two distributions can be quantified by **relative entropy**, or the Kullback-Leibler divergence (Kullback and Leibler, 1951). The Kullback-Leibler or **KL divergence** $D(P||Q)$ expresses the difference between two probability distributions P and Q

$$D(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \quad (20.38)$$

The selectional preference $S_R(v)$ uses the KL divergence to express how much information, in bits, the verb v expresses about the possible semantic class of its argument.

$$\begin{aligned} S_R(v) &= D(P(c|v)||P(c)) \\ &= \sum_c P(c|v) \log \frac{P(c|v)}{P(c)} \end{aligned} \quad (20.39)$$

selectional association Resnik then defines the **selectional association** of a particular class and verb as the relative contribution of that class to the general selectional preference of the verb:

$$A_R(v, c) = \frac{1}{S_R(v)} P(c|v) \log \frac{P(c|v)}{P(c)} \quad (20.40)$$

The selectional association is thus a probabilistic measure of the strength of association between a predicate and a class dominating the argument to the predicate. Resnik estimates the probabilities for these associations by parsing a corpus, counting all the times each predicate occurs with each argument word, and assuming that each word is a partial observation of all the WordNet concepts containing the word. The following table from [Resnik \(1996\)](#) shows some sample high and low selectional associations for verbs and some WordNet semantic classes of their direct objects.

Verb	Direct Object		Direct Object	
	Semantic Class	Assoc	Semantic Class	Assoc
read	WRITING	6.80	ACTIVITY	-.20
write	WRITING	7.26	COMMERCE	0
see	ENTITY	5.79	METHOD	-0.01

Selectional Preference via Conditional Probability

An alternative to using selectional association between a verb and the WordNet class of its arguments is to use the conditional probability of an argument word given a predicate verb, directly modeling the strength of association of one verb (predicate) with one noun (argument).

The conditional probability model can be computed by parsing a very large corpus (billions of words), and computing co-occurrence counts: how often a given verb occurs with a given noun in a given relation. The conditional probability of an argument noun given a verb for a particular relation $P(n|v, r)$ can then be used as a selectional preference metric for that pair of words ([Brockmann and Lapata 2003](#), [Keller and Lapata 2003](#)):

$$P(n|v, r) = \begin{cases} \frac{C(n, v, r)}{C(v, r)} & \text{if } C(n, v, r) > 0 \\ 0 & \text{otherwise} \end{cases}$$

The inverse probability $P(v|n, r)$ was found to have better performance in some cases ([Brockmann and Lapata, 2003](#)):

$$P(v|n, r) = \begin{cases} \frac{C(n, v, r)}{C(n, r)} & \text{if } C(n, v, r) > 0 \\ 0 & \text{otherwise} \end{cases}$$

An even simpler approach is to use the simple log co-occurrence frequency of the predicate with the argument $\log count(v, n, r)$ instead of conditional probability; this seems to do better for extracting preferences for syntactic subjects rather than objects ([Brockmann and Lapata, 2003](#)).

Evaluating Selectional Preferences

pseudowords

One way to evaluate models of selectional preferences is to use **pseudowords** ([Gale et al. 1992c](#), [Schütze 1992a](#)). A pseudoword is an artificial word created by concatenating a test word in some context (say *banana*) with a confounder word (say *door*)

to create *banana-door*). The task of the system is to identify which of the two words is the original word. To evaluate a selectional preference model (for example on the relationship between a verb and a direct object) we take a test corpus and select all verb tokens. For each verb token (say *drive*) we select the direct object (e.g., *car*), concatenated with a confounder word that is its *nearest neighbor*, the noun with the frequency closest to the original (say *house*), to make *car/house*). We then use the selectional preference model to choose which of *car* and *house* are more preferred objects of *drive*, and compute how often the model chooses the correct original object (e.g., *car*) (Chambers and Jurafsky, 2010).

Another evaluation metric is to get human preferences for a test set of verb-argument pairs, and have them rate their degree of plausibility. This is usually done by using magnitude estimation, a technique from psychophysics, in which subjects rate the plausibility of an argument proportional to a modulus item. A selectional preference model can then be evaluated by its correlation with the human preferences (Keller and Lapata, 2003).

20.8 Primitive Decomposition of Predicates

compositional analysis

One way of thinking about the semantic roles we have discussed through the chapter is that they help us define the roles that arguments play in a decompositional way, based on finite lists of thematic roles (agent, patient, instrument, proto-agent, protopatient, etc.). This idea of decomposing meaning into sets of primitive semantics elements or features, called **primitive decomposition** or **compositional analysis**, has been taken even further, and focused particularly on predicates.

Consider these examples of the verb *kill*:

(20.41) Jim killed his philodendron.

(20.42) Jim did something to cause his philodendron to become not alive.

There is a truth-conditional ('propositional semantics') perspective from which these two sentences have the same meaning. Assuming this equivalence, we could represent the meaning of *kill* as:

(20.43) $\text{KILL}(x,y) \Leftrightarrow \text{CAUSE}(x, \text{BECOME}(\text{NOT}(\text{ALIVE}(y))))$

thus using semantic primitives like *do*, *cause*, *become not*, and *alive*.

Indeed, one such set of potential semantic primitives has been used to account for some of the verbal alternations discussed in Section 20.2 (Lakoff 1965, Dowty 1979). Consider the following examples.

(20.44) John opened the door. $\Rightarrow \text{CAUSE}(\text{John}, \text{BECOME}(\text{OPEN}(\text{door})))$

(20.45) The door opened. $\Rightarrow \text{BECOME}(\text{OPEN}(\text{door}))$

(20.46) The door is open. $\Rightarrow \text{OPEN}(\text{door})$

The decompositional approach asserts that a single state-like predicate associated with *open* underlies all of these examples. The differences among the meanings of these examples arises from the combination of this single predicate with the primitives *CAUSE* and *BECOME*.

While this approach to primitive decomposition can explain the similarity between states and actions or causative and non-causative predicates, it still relies on having a large number of predicates like *open*. More radical approaches choose to

break down these predicates as well. One such approach to verbal predicate decomposition that played a role in early natural language understanding systems is **conceptual dependency** (CD), a set of ten primitive predicates, shown in Fig. 20.8.

Primitive	Definition
ATRANS	The abstract transfer of possession or control from one entity to another
PTRANS	The physical transfer of an object from one location to another
MTRANS	The transfer of mental concepts between entities or within an entity
MBUILD	The creation of new information within an entity
PROPEL	The application of physical force to move an object
MOVE	The integral movement of a body part by an animal
INGEST	The taking in of a substance by an animal
EXPTEL	The expulsion of something from an animal
SPEAK	The action of producing a sound
ATTEND	The action of focusing a sense organ

Figure 20.8 A set of conceptual dependency primitives.

Below is an example sentence along with its CD representation. The verb *brought* is translated into the two primitives ATRANS and PTRANS to indicate that the waiter both physically conveyed the check to Mary and passed control of it to her. Note that CD also associates a fixed set of thematic roles with each primitive to represent the various participants in the action.

(20.47) The waiter brought Mary the check.

$$\begin{aligned} \exists x, y \text{ Atrans}(x) \wedge \text{Actor}(x, \text{Waiter}) \wedge \text{Object}(x, \text{Check}) \wedge \text{To}(x, \text{Mary}) \\ \wedge \text{Ptrans}(y) \wedge \text{Actor}(y, \text{Waiter}) \wedge \text{Object}(y, \text{Check}) \wedge \text{To}(y, \text{Mary}) \end{aligned}$$

20.9 Summary

- **Semantic roles** are abstract models of the role an argument plays in the event described by the predicate.
- **Thematic roles** are a model of semantic roles based on a single finite list of roles. Other semantic role models include per-verb semantic role lists and **proto-agent/proto-patient**, both of which are implemented in **PropBank**, and per-frame role lists, implemented in **FrameNet**.
- **Semantic role labeling** is the task of assigning semantic role labels to the constituents of a sentence. The task is generally treated as a supervised machine learning task, with models trained on PropBank or FrameNet. Algorithms generally start by parsing a sentence and then automatically tag each parse tree node with a semantic role. Neural models map straight from words end-to-end.
- Semantic **selectional restrictions** allow words (particularly predicates) to post constraints on the semantic properties of their argument words. **Selectional**

preference models (like **selectional association** or simple conditional probability) allow a weight or probability to be assigned to the association between a predicate and an argument word or class.

Bibliographical and Historical Notes

Although the idea of semantic roles dates back to Pāṇini, they were re-introduced into modern linguistics by [Gruber \(1965\)](#), [Fillmore \(1966\)](#) and [Fillmore \(1968\)](#). Fillmore, interestingly, had become interested in argument structure by studying Lucien Tesnière's groundbreaking *Éléments de Syntaxe Structurale* ([Tesnière, 1959](#)) in which the term 'dependency' was introduced and the foundations were laid for dependency grammar. Following Tesnière's terminology, Fillmore first referred to argument roles as *actants* ([Fillmore, 1966](#)) but quickly switched to the term *case*, (see [Fillmore \(2003\)](#)) and proposed a universal list of semantic roles or cases (Agent, Patient, Instrument, etc.), that could be taken on by the arguments of predicates. Verbs would be listed in the lexicon with their **case frame**, the list of obligatory (or optional) case arguments.

The idea that semantic roles could provide an intermediate level of semantic representation that could help map from syntactic parse structures to deeper, more fully-specified representations of meaning was quickly adopted in natural language processing, and systems for extracting case frames were created for machine translation ([Wilks, 1973](#)), question-answering ([Hendrix et al., 1973](#)), spoken-language understanding ([Nash-Webber, 1975](#)), and dialogue systems ([Bobrow et al., 1977](#)). General-purpose semantic role labelers were developed. The earliest ones ([Simmons, 1973](#)) first parsed a sentence by means of an ATN (Augmented Transition Network) parser. Each verb then had a set of rules specifying how the parse should be mapped to semantic roles. These rules mainly made reference to grammatical functions (subject, object, complement of specific prepositions) but also checked constituent internal features such as the animacy of head nouns. Later systems assigned roles from pre-built parse trees, again by using dictionaries with verb-specific case frames ([Levin 1977, Marcus 1980](#)).

By 1977 case representation was widely used and taught in AI and NLP courses, and was described as a standard of natural language understanding in the first edition of Winston's (1977) textbook *Artificial Intelligence*.

In the 1980s Fillmore proposed his model of *frame semantics*, later describing the intuition as follows:

“The idea behind frame semantics is that speakers are aware of possibly quite complex situation types, packages of connected expectations, that go by various names—frames, schemas, scenarios, scripts, cultural narratives, memes—and the words in our language are understood with such frames as their presupposed background.” ([Fillmore, 2012, p. 712](#))

The word *frame* seemed to be in the air for a suite of related notions proposed at about the same time by [Minsky \(1974\)](#), [Hymes \(1974\)](#), and [Goffman \(1974\)](#), as well as related notions with other names like *scripts* ([Schank and Abelson, 1975](#)) and *schemata* ([Bobrow and Norman, 1975](#)) (see [Tannen \(1979\)](#) for a comparison). Fillmore was also influenced by the semantic field theorists and by a visit to the Yale AI lab where he took notice of the lists of slots and fillers used by early information

extraction systems like DeJong (1982) and Schank and Abelson (1977). In the 1990s Fillmore drew on these insights to begin the FrameNet corpus annotation project.

At the same time, Beth Levin drew on her early case frame dictionaries (Levin, 1977) to develop her book which summarized sets of verb classes defined by shared argument realizations (Levin, 1993). The VerbNet project built on this work (Kipper et al., 2000), leading soon afterwards to the PropBank semantic-role-labeled corpus created by Martha Palmer and colleagues (Palmer et al., 2005).

The combination of rich linguistic annotation and corpus-based approach instantiated in FrameNet and PropBank led to a revival of automatic approaches to semantic role labeling, first on FrameNet (Gildea and Jurafsky, 2000) and then on PropBank data (Gildea and Palmer, 2002, *inter alia*). The problem first addressed in the 1970s by handwritten rules was thus now generally recast as one of supervised machine learning enabled by large and consistent databases. Many popular features used for role labeling are defined in Gildea and Jurafsky (2002), Surdeanu et al. (2003), Xue and Palmer (2004), Pradhan et al. (2005), Che et al. (2009), and Zhao et al. (2009). The use of dependency rather than constituency parses was introduced in the CoNLL-2008 shared task (Surdeanu et al., 2008b). For surveys see Palmer et al. (2010) and Márquez et al. (2008).

The use of neural approaches to semantic role labeling was pioneered by Collobert et al. (2011), who applied a CRF on top of a convolutional net. Early work like Foland, Jr. and Martin (2015) focused on using dependency features. Later work eschewed syntactic features altogether; (Zhou and Xu, 2015b) introduced the use of a stacked (6-8 layer) bi-LSTM architecture, and (He et al., 2017) showed how to augment the bi-LSTM architecture with highway networks and also replace the CRF with A* decoding that make it possible to apply a wide variety of global constraints in SRL decoding.

implicit argument

Most semantic role labeling schemes only work within a single sentence, focusing on the object of the verbal (or nominal, in the case of NomBank) predicate. However, in many cases, a verbal or nominal predicate may have an **implicit argument**: one that appears only in a contextual sentence, or perhaps not at all and must be inferred. In the two sentences *This house has a new owner. The sale was finalized 10 days ago.* the *sale* in the second sentence has no ARG1, but a reasonable reader would infer that the Arg1 should be the *house* mentioned in the prior sentence. Finding these arguments, **implicit argument detection** (sometimes shortened as **iSRL**) was introduced by Gerber and Chai (2010) and Ruppenhofer et al. (2010). See Do et al. (2017) for more recent neural models.

iSRL

To avoid the need for huge labeled training sets, unsupervised approaches for semantic role labeling attempt to induce the set of semantic roles by clustering over arguments. The task was pioneered by Riloff and Schmelzenbach (1998) and Swier and Stevenson (2004); see Grenager and Manning (2006), Titov and Klementiev (2012), Lang and Lapata (2014), Woodsend and Lapata (2015), and Titov and Khodam (2014).

Recent innovations in frame labeling include **connotation frames**, which mark richer information about the argument of predicates. Connotation frames mark the sentiment of the writer or reader toward the arguments (for example using the verb *survive* in *he survived a bombing* expresses the writer's sympathy toward the subject *he* and negative sentiment toward the bombing. Connotation frames also mark *effect* (something bad happened to x), *value*: (x is valuable), and *mental state*: (x is distressed by the event) (Rashkin et al. 2016, Rashkin et al. 2017). Connotation frames can also mark the *power differential* between the arguments (using the verb *implore*

means that the theme argument has greater power than the agent), and the *agency* of each argument (*waited* is low agency). Fig. 20.9 shows a visualization from Sap et al. (2017).

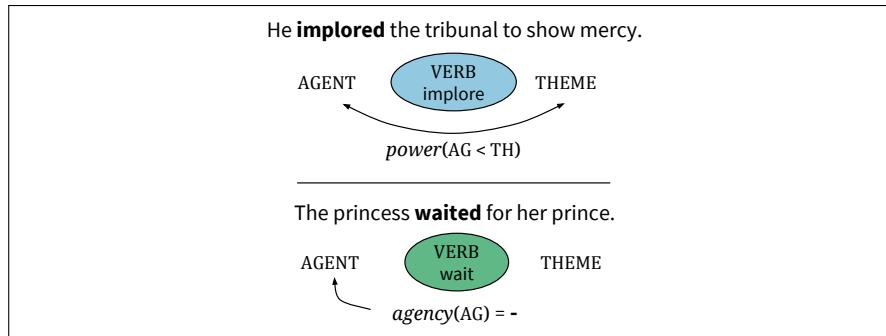


Figure 20.9 The connotation frames of Sap et al. (2017), showing that the verb *implore* implies the agent has lower power than the theme (in contrast, say, with a verb like *demanded*), and showing the low level of agency of the subject of *waited*. Figure from Sap et al. (2017).

Selectional preference has been widely studied beyond the selectional association models of Resnik (1993) and Resnik (1996). Methods have included clustering (Rooth et al., 1999), discriminative learning (Bergsma et al., 2008a), and topic models (Séaghdha 2010, Ritter et al. 2010), and constraints can be expressed at the level of words or classes (Agirre and Martínez, 2001). Selectional preferences have also been successfully integrated into semantic role labeling (Erk 2007, Zapirain et al. 2013, Do et al. 2017).

Exercises

CHAPTER

21

Lexicons for Sentiment, Affect,
and Connotation

“[W]e write, not with the fingers, but with the whole person. The nerve which controls the pen winds itself about every fibre of our being, threads the heart, pierces the liver.”

Virginia Woolf, *Orlando*

“She runs the gamut of emotions from A to B.”

Dorothy Parker, reviewing Hepburn’s performance in *Little Women*

“Festerling’s always bad. There’s no good kind of festering.”

Adventure Time, Season 5

affective

In this chapter we turn to tools for interpreting **affective** meaning, extending our study of sentiment analysis in Chapter 4. We use the word ‘affective’, following the tradition in *affective computing* (Picard, 1995) to mean emotion, sentiment, personality, mood, and attitudes. Affective meaning is closely related to **subjectivity**, the study of a speaker or writer’s evaluations, opinions, emotions, and speculations (Wiebe et al., 1999).

subjectivity

How should affective meaning be defined? One influential typology of affective states comes from Scherer (2000), who defines each class of affective states by factors like its cognitive realization and time course:

We can design extractors for each of these kinds of affective states. Chapter 4 already introduced *sentiment analysis*, the task of extracting the positive or negative orientation that a writer expresses in a text. This corresponds in Scherer’s typology to the extraction of **attitudes**: figuring out what people like or dislike, from affect-rich texts like consumer reviews of books or movies, newspaper editorials, or public sentiment in blogs or tweets.

Detecting **emotion** and **moods** is useful for detecting whether a student is confused, engaged, or certain when interacting with a tutorial system, whether a caller to a help line is frustrated, whether someone’s blog posts or tweets indicated depression. Detecting emotions like fear in novels, for example, could help us trace what groups or situations are feared and how that changes over time.

Detecting different **interpersonal stances** can be useful when extracting information from human-human conversations. The goal here is to detect stances like friendliness or awkwardness in interviews or friendly conversations, for example for summarizing meetings or finding parts of a conversation where people are especially excited or engaged, conversational **hot spots** that can help in meeting summarization. Detecting the **personality** of a user—such as whether the user is an **extrovert** or the extent to which they are **open to experience**—can help improve conversa-

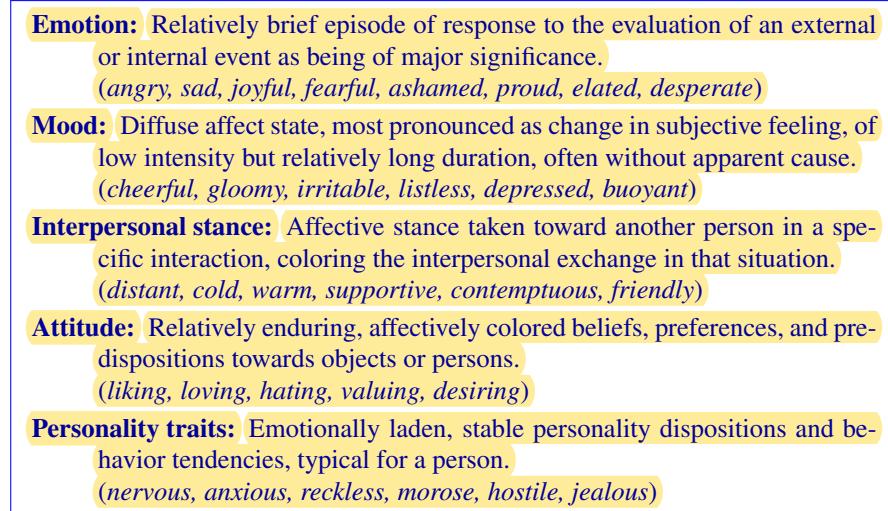


Figure 21.1 The Scherer typology of affective states (Scherer, 2000).

tional agents, which seem to work better if they match users' personality expectations (Mairesse and Walker, 2008). And affect is important for generation as well as recognition; synthesizing affect is important for conversational agents in various domains, including literacy tutors such as children's storybooks, or computer games.

In Chapter 4 we introduced the use of naive Bayes classification to classify a document's sentiment. Various classifiers have been successfully applied to many of these tasks, using all the words in the training set as input to a classifier which then determines the affect status of the text.

connotations

In this chapter we focus on an alternative model, in which instead of using every word as a feature, we focus only on certain words, ones that carry particularly strong cues to affect or sentiment. We call these lists of words **affective lexicons** or **sentiment lexicons**. These lexicons presuppose a fact about semantics: that words have *affective meanings* or **connotations**. The word *connotation* has different meanings in different fields, but here we use it to mean the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations. In addition to their ability to help determine the affective status of a text, connotation lexicons can be useful features for other kinds of affective tasks, and for computational social science analysis.

In the next sections we introduce basic theories of emotion, show how sentiment lexicons are a special case of emotion lexicons, and mention some useful lexicons. We then survey three ways for building lexicons: human labeling, semi-supervised, and supervised. Finally, we turn to other kinds of affective meaning like personality, stance, and entity-centric affect, and introduce connotation frames.

21.1 Defining Emotion

emotion One of the most important affective classes is **emotion**, which Scherer (2000) defines as a "relatively brief episode of response to the evaluation of an external or internal event as being of major significance".

Detecting emotion has the potential to improve a number of language processing

basic emotions

tasks. Automatically detecting emotions in reviews or customer responses (anger, dissatisfaction, trust) could help businesses recognize specific problem areas or ones that are going well. Emotion recognition could help dialog systems like tutoring systems detect that a student was unhappy, bored, hesitant, confident, and so on. Emotion can play a role in medical informatics tasks like detecting depression or suicidal intent. Detecting emotions expressed toward characters in novels might play a role in understanding how different social groups were viewed by society at different times.

There are two widely-held families of theories of emotion. In one family, emotions are viewed as fixed atomic units, limited in number, and from which others are generated, often called **basic emotions** (Tomkins 1962, Plutchik 1962). Perhaps most well-known of this family of theories are the 6 emotions proposed by Ekman (see for example Ekman 1999) as a set of emotions that is likely to be universally present in all cultures: *surprise, happiness, anger, fear, disgust, sadness*. Another atomic theory is the Plutchik (1980) wheel of emotion, consisting of 8 basic emotions in four opposing pairs: *joy–sadness, anger–fear, trust–disgust, and anticipation–surprise*, together with the emotions derived from them, shown in Fig. 21.2.

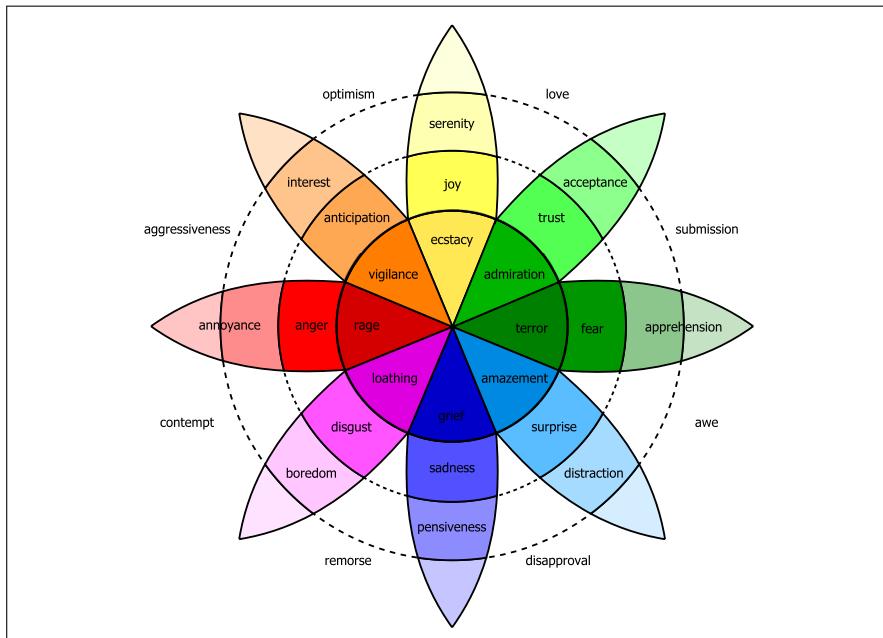


Figure 21.2 Plutchik wheel of emotion.

The second class of emotion theories views emotion as a space in 2 or 3 dimensions (Russell, 1980). Most models include the two dimensions **valence** and **arousal**, and many add a third, **dominance**. These can be defined as:

valence: the pleasantness of the stimulus

arousal: the intensity of emotion provoked by the stimulus

dominance: the degree of control exerted by the stimulus

In the next sections we'll see lexicons for both kinds of theories of emotion.

Sentiment can be viewed as a special case of this second view of emotions as points in space. In particular, the **valence** dimension, measuring how pleasant or unpleasant a word is, is often used directly as a measure of sentiment.

21.2 Available Sentiment and Affect Lexicons

A wide variety of affect lexicons have been created and released. The most basic lexicons label words along one dimension of semantic variability, generally called “sentiment” or “valence”.

General Inquirer

In the simplest lexicons this dimension is represented in a binary fashion, with a wordlist for positive words and a wordlist for negative words. The oldest is the **General Inquirer** (Stone et al., 1966), which drew on content analysis and on early work in the cognitive psychology of word meaning (Osgood et al., 1957). The General Inquirer has a lexicon of 1915 positive words and a lexicon of 2291 negative words (as well as other lexicons discussed below). The MPQA Subjectivity lexicon (Wilson et al., 2005) has 2718 positive and 4912 negative words drawn from prior lexicons plus a bootstrapped list of subjective words and phrases (Riloff and Wiebe, 2003). Each entry in the lexicon is hand-labeled for sentiment and also labeled for reliability (strongly subjective or weakly subjective). The polarity lexicon of Hu and Liu (2004b) gives 2006 positive and 4783 negative words, drawn from product reviews, labeled using a bootstrapping method from WordNet.

Positive	admire, amazing, assure, celebration, charm, eager, enthusiastic, excellent, fancy, fantastic, frolic, graceful, happy, joy, luck, majesty, mercy, nice, patience, perfect, proud, rejoice, relief, respect, satisfactorily, sensational, super, terrific, thank, vivid, wise, wonderful, zest
Negative	abominable, anger, anxious, bad, catastrophe, cheap, complaint, condescending, deceit, defective, disappointment, embarrass, fake, fear, filthy, fool, guilt, hate, idiot, inflict, lazy, miserable, mourn, nervous, objection, pest, plot, reject, scream, silly, terrible, unfriendly, vile, wicked

Figure 21.3 Some samples of words with consistent sentiment across three sentiment lexicons: the General Inquirer (Stone et al., 1966), the MPQA Subjectivity lexicon (Wilson et al., 2005), and the polarity lexicon of Hu and Liu (2004b).

Slightly more general than these sentiment lexicons are lexicons that assign each word a value on all three affective dimensions. The NRC Valence, Arousal, and Dominance (VAD) lexicon (Mohammad, 2018a) assigns valence, arousal, and dominance scores to 20,000 words. Some examples are shown in Fig. 21.4.

	Valence		Arousal		Dominance
vacation	.840	enraged	.962	powerful	.991
delightful	.918	party	.840	authority	.935
whistle	.653	organized	.337	saxophone	.482
consolation	.408	effortless	.120	discouraged	.0090
torture	.115	napping	.046	weak	.045

Figure 21.4 Samples of the values of selected words on the three emotional dimensions from Mohammad (2018a).

EmoLex

The NRC Word-Emotion Association Lexicon, also called **EmoLex** (Mohammad and Turney, 2013), uses the Plutchik (1980) 8 basic emotions defined above. The lexicon includes around 14,000 words including words from prior lexicons as well as frequent nouns, verbs, adverbs and adjectives. Values from the lexicon for some sample words:

Word	anger	anticipation	disgust	fear	joy	sadness	surprise	trust	positive	negative
reward	0	1	0	0	1	0	1	1	1	0
worry	0	1	0	1	0	1	0	0	0	1
tenderness	0	0	0	0	1	0	0	0	1	0
sweetheart	0	1	0	0	1	1	0	1	1	0
suddenly	0	0	0	0	0	0	1	0	0	0
thirst	0	1	0	0	0	1	1	0	0	0
garbage	0	0	1	0	0	0	0	0	0	1

For a smaller set of 5,814 words, the NRC Emotion/Affect Intensity Lexicon ([Mohammad, 2018b](#)) contains real-valued scores of association for anger, fear, joy, and sadness; Fig. 21.5 shows examples.

	Anger	Fear	Joy	Sadness	
outraged	0.964	horror	0.923	superb	0.864
violence	0.742	anguish	0.703	cheered	0.773
coup	0.578	pestilence	0.625	rainbow	0.531
oust	0.484	stressed	0.531	gesture	0.387
suspicious	0.484	failing	0.531	warms	0.391
nurture	0.059	confident	0.094	hardship	.031
				sing	0.017

Figure 21.5 Sample emotional intensities for words for anger, fear, joy, and sadness from [Mohammad \(2018b\)](#).

LIWC

LIWC, Linguistic Inquiry and Word Count, is a widely used set of 73 lexicons containing over 2300 words ([Pennebaker et al., 2007](#)), designed to capture aspects of lexical meaning relevant for social psychological tasks. In addition to sentiment-related lexicons like ones for negative emotion (*bad, weird, hate, problem, tough*) and positive emotion (*love, nice, sweet*), LIWC includes lexicons for categories like anger, sadness, cognitive mechanisms, perception, tentative, and inhibition, shown in Fig. 21.6.

There are various other hand-built affective lexicons. The General Inquirer includes additional lexicons for dimensions like strong vs. weak, active vs. passive, overstated vs. understated, as well as lexicons for categories like pleasure, pain, virtue, vice, motivation, and cognitive orientation.

concrete abstract

Another useful feature for various tasks is the distinction between **concrete** words like *banana* or *bathrobe* and **abstract** words like *belief* and *although*. The lexicon in [Brysbaert et al. \(2014\)](#) used crowdsourcing to assign a rating from 1 to 5 of the concreteness of 40,000 words, thus assigning *banana, bathrobe*, and *bagel* 5, *belief* 1.19, *although* 1.07, and in between words like *brisk* a 2.5.

21.3 Creating Affect Lexicons by Human Labeling

crowdsourcing

The earliest method used to build affect lexicons, and still in common use, is to have humans label each word. This is now most commonly done via **crowdsourcing**: breaking the task into small pieces and distributing them to a large number of anno-

Positive Emotion	Negative Emotion	Insight	Inhibition	Family	Negate
appreciat*	anger*	aware*	avoid*	brother*	aren't
comfort*	bore*	believe	careful*	cousin*	cannot
great	cry	decid*	hesitat*	daughter*	didn't
happy	despair*	feel	limit*	family	neither
interest	fail*	figur*	oppos*	father*	never
joy*	fear	know	prevent*	grandf*	no
perfect*	griev*	knew	reluctan*	grandm*	nobod*
please*	hate*	means	safe*	husband	none
safe*	panic*	notice*	stop	mom	nor
terrific	suffers	recogni*	stubborn*	mother	nothing
value	terrify	sense	wait	niece*	nowhere
wow*	violent*	think	wary	wife	without

Figure 21.6 Samples from 5 of the 73 lexical categories in LIWC ([Pennebaker et al., 2007](#)).

The * means the previous letters are a word prefix and all words with that prefix are included in the category.

tators. Let's take a look at some of the methodological choices for two crowdsourced emotion lexicons.

The NRC Emotion Lexicon (EmoLex) ([Mohammad and Turney, 2013](#)), labeled emotions in two steps. To ensure that the annotators were judging the correct sense of the word, they first answered a multiple-choice synonym question that primed the correct sense of the word (without requiring the annotator to read a potentially confusing sense definition). These were created automatically using the headwords associated with the thesaurus category of the sense in question in the Macquarie dictionary and the headwords of 3 random distractor categories. An example:

Which word is closest in meaning (most related) to startle?

- automobile
- shake
- honesty
- entertain

For each word (e.g. *startle*), the annotator was then asked to rate how associated that word is with each of the 8 emotions (*joy, fear, anger*, etc.). The associations were rated on a scale of *not, weakly, moderately, and strongly* associated. Outlier ratings were removed, and then each term was assigned the class chosen by the majority of the annotators, with ties broken by choosing the stronger intensity, and then the 4 levels were mapped into a binary label for each word (no and weak mapped to 0, moderate and strong mapped to 1).

best-worst scaling

The NRC VAD Lexicon ([Mohammad, 2018a](#)) was built by selecting words and emoticons from prior lexicons and annotating them with crowd-sourcing using **best-worst scaling** ([Louviere et al. 2015, Kiritchenko and Mohammad 2017](#)). In best-worst scaling, annotators are given N items (usually 4) and are asked which item is the **best** (highest) and which is the **worst** (lowest) in terms of some property. The set of words used to describe the ends of the scales are taken from prior literature. For valence, for example, the raters were asked:

Q1. Which of the four words below is associated with the MOST happiness / pleasure / positiveness / satisfaction / contentedness / hopefulness
OR LEAST unhappiness / annoyance / negativeness / dissatisfaction /

melancholy / despair? (Four words listed as options.)

Q2. Which of the four words below is associated with the LEAST happiness / pleasure / positiveness / satisfaction / contentedness / hopefulness OR MOST unhappiness / annoyance / negativeness / dissatisfaction / melancholy / despair? (Four words listed as options.)

split-half reliability

The score for each word in the lexicon is the proportion of times the item was chosen as the best (highest V/A/D) minus the proportion of times the item was chosen as the worst (lowest V/A/D). The agreement between annotations are evaluated by **split-half reliability**: split the corpus in half and compute the correlations between the annotations in the two halves.

21.4 Semi-supervised Induction of Affect Lexicons

Another common way to learn sentiment lexicons is to start from a set of seed words that define two poles of a semantic axis (words like *good* or *bad*), and then find ways to label each word w by its similarity to the two seed sets. Here we summarize two families of seed-based semi-supervised lexicon induction algorithms, axis-based and graph-based.

21.4.1 Semantic Axis Methods

One of the most well-known lexicon induction methods, the [Turney and Littman \(2003\)](#) algorithm, is given seed words like *good* or *bad*, and then for each word w to be labeled, measures both how similar it is to *good* and how different it is from *bad*. Here we describe a slight extension of the algorithm due to [An et al. \(2018\)](#), which is based on computing a **semantic axis**.

In the first step, we choose seed words by hand. There are two methods for dealing with the fact that the affect of a word is different in different contexts: (1) start with a single large seed lexicon and rely on the induction algorithm to fine-tune it to the domain, or (2) choose different seed words for different genres. [Hellrich et al. \(2019\)](#) suggests that for modeling affect across different historical time periods, starting with a large modern affect dictionary is better than small seedsets tuned to be stable across time. As an example of the second approach, [Hamilton et al. \(2016a\)](#) define one set of seed words for general sentiment analysis, a different set for Twitter, and yet another set for sentiment in financial text:

Domain	Positive seeds	Negative seeds
General	good, lovely, excellent, fortunate, pleasant, delightful, perfect, loved, love, happy	bad, horrible, poor, unfortunate, unpleasant, disgusting, evil, hated, hate, unhappy
Twitter	love, loved, loves, awesome, nice, amazing, best, fantastic, correct, happy	hate, hated, hates, terrible, nasty, awful, worst, horrible, wrong, sad
Finance	successful, excellent, profit, beneficial, improving, improved, success, gains, positive	negligent, loss, volatile, wrong, losses, damages, bad, litigation, failure, down, negative

In the second step, we compute embeddings for each of the pole words. These embeddings can be off-the-shelf word2vec embeddings, or can be computed directly

on a specific corpus (for example using a financial corpus if a finance lexicon is the goal), or we can fine-tune off-the-shelf embeddings to a corpus. Fine-tuning is especially important if we have a very specific genre of text but don't have enough data to train good embeddings. In fine-tuning, we begin with off-the-shelf embeddings like word2vec, and continue training them on the small target corpus.

Once we have embeddings for each pole word, we create an embedding that represents each pole by taking the centroid of the embeddings of each of the seed words; recall that the centroid is the multidimensional version of the mean. Given a set of embeddings for the positive seed words $S^+ = \{E(w_1^+), E(w_2^+), \dots, E(w_n^+)\}$, and embeddings for the negative seed words $S^- = \{E(w_1^-), E(w_2^-), \dots, E(w_m^-)\}$, the pole centroids are:

$$\begin{aligned}\mathbf{v}^+ &= \frac{1}{n} \sum_{i=1}^n E(w_i^+) \\ \mathbf{v}^- &= \frac{1}{m} \sum_{i=1}^m E(w_i^-)\end{aligned}\tag{21.1}$$

The semantic axis defined by the poles is computed just by subtracting the two vectors:

$$\mathbf{v}_{axis} = \mathbf{v}^+ - \mathbf{v}^-\tag{21.2}$$

\mathbf{v}_{axis} , the semantic axis, is a vector in the direction of sentiment. Finally, we compute how close each word w is to this sentiment axis, by taking the cosine between w 's embedding and the axis vector. A higher cosine means that w is more aligned with S^+ than S^- .

$$\begin{aligned}\text{score}(w) &= (\cos(E(w), \mathbf{v}_{axis})) \\ &= \frac{E(w) \cdot \mathbf{v}_{axis}}{\|E(w)\| \|\mathbf{v}_{axis}\|}\end{aligned}\tag{21.3}$$

If a dictionary of words with sentiment scores is sufficient, we're done! Or if we need to group words into a positive and a negative lexicon, we can use a threshold or other method to give us discrete lexicons.

21.4.2 Label Propagation

An alternative family of methods defines lexicons by propagating sentiment labels on graphs, an idea suggested in early work by [Hatzivassiloglou and McKeown \(1997\)](#). We'll describe the simple SentProp (Sentiment Propagation) algorithm of [Hamilton et al. \(2016a\)](#), which has four steps:

1. **Define a graph:** Given word embeddings, build a weighted lexical graph by connecting each word with its k nearest neighbors (according to cosine-similarity). The weights of the edge between words w_i and w_j are set as:

$$\mathbf{E}_{i,j} = \arccos \left(-\frac{\mathbf{w}_i^\top \mathbf{w}_j}{\|\mathbf{w}_i\| \|\mathbf{w}_j\|} \right).\tag{21.4}$$

2. **Define a seed set:** Choose positive and negative seed words.
3. **Propagate polarities from the seed set:** Now we perform a random walk on this graph, starting at the seed set. In a random walk, we start at a node and

then choose a node to move to with probability proportional to the edge probability. A word's polarity score for a seed set is proportional to the probability of a random walk from the seed set landing on that word, (Fig. 21.7).

4. **Create word scores:** We walk from both positive and negative seed sets, resulting in positive ($\text{score}^+(w_i)$) and negative ($\text{score}^-(w_i)$) label scores. We then combine these values into a positive-polarity score as:

$$\text{score}^+(w_i) = \frac{\text{score}^+(w_i)}{\text{score}^+(w_i) + \text{score}^-(w_i)} \quad (21.5)$$

It's often helpful to standardize the scores to have zero mean and unit variance within a corpus.

5. **Assign confidence to each score:** Because sentiment scores are influenced by the seed set, we'd like to know how much the score of a word would change if a different seed set is used. We can use bootstrap-sampling to get confidence regions, by computing the propagation B times over random subsets of the positive and negative seed sets (for example using $B = 50$ and choosing 7 of the 10 seed words each time). The standard deviation of the bootstrap-sampled polarity scores gives a confidence measure.



Figure 21.7 Intuition of the SENTPROP algorithm. (a) Run random walks from the seed words. (b) Assign polarity scores (shown here as colors green or red) based on the frequency of random walk visits.

21.4.3 Other Methods

The core of semisupervised algorithms is the metric for measuring similarity with the seed words. The Turney and Littman (2003) and Hamilton et al. (2016a) approaches above used embedding cosine as the distance metric: words were labeled as positive basically if their embeddings had high cosines with positive seeds and low cosines with negative seeds. Other methods have chosen other kinds of distance metrics besides embedding cosine.

For example the Hatzivassiloglou and McKeown (1997) algorithm uses syntactic cues; two adjectives are considered similar if they were frequently conjoined by *and* and rarely conjoined by *but*. This is based on the intuition that adjectives conjoined by the words *and* tend to have the same polarity; positive adjectives are generally coordinated with positive, negative with negative:

fair and legitimate, corrupt and brutal

but less often positive adjectives coordinated with negative:

*fair and brutal, *corrupt and legitimate

By contrast, adjectives conjoined by *but* are likely to be of opposite polarity:

fair but brutal

Another cue to opposite polarity comes from morphological negation (*un-*, *im-*, *-less*). Adjectives with the same root but differing in a morphological negative (*adequate/inadequate, thoughtful/thoughtless*) tend to be of opposite polarity.

Yet another method for finding words that have a similar polarity to seed words is to make use of a thesaurus like WordNet (Kim and Hovy 2004, Hu and Liu 2004b). A word's synonyms presumably share its polarity while a word's antonyms probably have the opposite polarity. After a seed lexicon is built, each lexicon is updated as follows, possibly iterated.

Lex⁺: Add synonyms of positive words (*well*) and antonyms (like *fine*) of negative words

Lex⁻: Add synonyms of negative words (*awful*) and antonyms (like *evil*) of positive words

An extension of this algorithm assigns polarity to WordNet senses, called **SentiWordNet** (Baccianella et al., 2010). Fig. 21.8 shows some examples.

Synset		Pos	Neg	Obj
good#6	'agreeable or pleasing'	1	0	0
respectable#2 honorable#4 good#4 estimable#2	'deserving of esteem'	0.75	0	0.25
estimable#3 computable#1	'may be computed or estimated'	0	0	1
sting#1 burn#4 bite#2	'cause a sharp or stinging pain'	0	0.875	.125
acute#6	'of critical importance and consequence'	0.625	0.125	.250
acute#4	'of an angle; less than 90 degrees'	0	0	1
acute#1	'having or experiencing a rapid onset and short but severe course'	0	0.5	0.5

Figure 21.8 Examples from SentiWordNet 3.0 (Baccianella et al., 2010). Note the differences between senses of homonymous words: *estimable*#3 is purely objective, while *estimable*#2 is positive; *acute* can be positive (*acute*#6), negative (*acute*#1), or neutral (*acute*#4).

In this algorithm, polarity is assigned to entire synsets rather than words. A positive lexicon is built from all the synsets associated with 7 positive words, and a negative lexicon from synsets associated with 7 negative words. A classifier is then trained from this data to take a WordNet gloss and decide if the sense being defined is positive, negative or neutral. A further step (involving a random-walk algorithm) assigns a score to each WordNet synset for its degree of positivity, negativity, and neutrality.

In summary, semisupervised algorithms use a human-defined set of seed words for the two poles of a dimension, and use similarity metrics like embedding cosine, coordination, morphology, or thesaurus structure to score words by how similar they are to the positive seeds and how dissimilar to the negative seeds.

21.5 Supervised Learning of Word Sentiment

Semi-supervised methods require only minimal human supervision (in the form of seed sets). But sometimes a supervision signal exists in the world and can be made use of. One such signal is the scores associated with *online reviews*.

The web contains an enormous number of online reviews for restaurants, movies, books, or other products, each of which have the text of the review along with an

Movie review excerpts (IMDb)	
10	A great movie. This film is just a wonderful experience. It's surreal, zany, witty and slapstick all at the same time. And terrific performances too.
1	This was probably the worst movie I have ever seen. The story went nowhere even though they could have done some interesting stuff with it.
Restaurant review excerpts (Yelp)	
5	The service was impeccable. The food was cooked and seasoned perfectly... The watermelon was perfectly square ... The grilled octopus was ... mouthwatering...
2	...it took a while to get our waters, we got our entree before our starter, and we never received silverware or napkins until we requested them...
Book review excerpts (GoodReads)	
1	I am going to try and stop being deceived by eye-catching titles. I so wanted to like this book and was so disappointed by it.
5	This book is hilarious. I would recommend it to anyone looking for a satirical read with a romantic twist and a narrator that keeps butting in
Product review excerpts (Amazon)	
5	The lid on this blender though is probably what I like the best about it... enables you to pour into something without even taking the lid off! ... the perfect pitcher! ... works fantastic.
1	I hate this blender... It is nearly impossible to get frozen fruit and ice to turn into a smoothie... You have to add a TON of liquid. I also wish it had a spout ...

Figure 21.9 Excerpts from some reviews from various review websites, all on a scale of 1 to 5 stars except IMDb, which is on a scale of 1 to 10 stars.

associated review score: a value that may range from 1 star to 5 stars, or scoring 1 to 10. Fig. 21.9 shows samples extracted from restaurant, book, and movie reviews.

We can use this review score as supervision: positive words are more likely to appear in 5-star reviews; negative words in 1-star reviews. And instead of just a binary polarity, this kind of supervision allows us to assign a word a more complex representation of its polarity: its distribution over stars (or other scores).

Thus in a ten-star system we could represent the sentiment of each word as a 10-tuple, each number a score representing the word's association with that polarity level. This association can be a raw count, or a likelihood $P(w|c)$, or some other function of the count, for each class c from 1 to 10.

For example, we could compute the IMDb likelihood of a word like *disappoint(ed/ing)* occurring in a 1 star review by dividing the number of times *disappoint(ed/ing)* occurs in 1-star reviews in the IMDb dataset (8,557) by the total number of words occurring in 1-star reviews (25,395,214), so the IMDb estimate of $P(disappointing|1)$ is .0003.

A slight modification of this weighting, the normalized likelihood, can be used as an illuminating visualization (Potts, 2011)¹:

$$\begin{aligned} P(w|c) &= \frac{\text{count}(w,c)}{\sum_{w \in c} \text{count}(w,c)} \\ \text{PottsScore}(w) &= \frac{P(w|c)}{\sum_c P(w|c)} \end{aligned} \quad (21.6)$$

Dividing the IMDb estimate $P(disappointing|1)$ of .0003 by the sum of the likelihood $P(w|c)$ over all categories gives a Potts score of 0.10. The word *disappointing*

¹ Potts shows that the normalized likelihood is an estimate of the posterior $P(c|w)$ if we make the incorrect but simplifying assumption that all categories c have equal probability.

Potts diagram thus is associated with the vector [.10, .12, .14, .14, .13, .11, .08, .06, .06, .05]. The **Potts diagram** (Potts, 2011) is a visualization of these word scores, representing the prior sentiment of a word as a distribution over the rating categories.

Fig. 21.10 shows the Potts diagrams for 3 positive and 3 negative scalar adjectives. Note that the curve for strongly positive scalars have the shape of the letter J, while strongly negative scalars look like a reverse J. By contrast, weakly positive and negative scalars have a hump-shape, with the maximum either below the mean (weakly negative words like *disappointing*) or above the mean (weakly positive words like *good*). These shapes offer an illuminating typology of affective meaning.



Figure 21.10 Potts diagrams (Potts, 2011) for positive and negative scalar adjectives, showing the J-shape and reverse J-shape for strongly positive and negative adjectives, and the hump-shape for more weakly polarized adjectives.

Fig. 21.11 shows the Potts diagrams for emphasizing and attenuating adverbs. Note that emphatics tend to have a J-shape (most likely to occur in the most positive reviews) or a U-shape (most likely to occur in the strongly positive and negative). Attenuators all have the hump-shape, emphasizing the middle of the scale and downplaying both extremes. The diagrams can be used both as a typology of lexical sentiment, and also play a role in modeling sentiment compositionality.

In addition to functions like posterior $P(c|w)$, likelihood $P(w|c)$, or normalized likelihood (Eq. 21.6) many other functions of the count of a word occurring with a sentiment label have been used. We'll introduce some of these on page 410, including ideas like normalizing the counts per writer in Eq. 21.14.

21.5.1 Log Odds Ratio Informative Dirichlet Prior

One thing we often want to do with word polarity is to distinguish between words that are more likely to be used in one category of texts than in another. We may, for example, want to know the words most associated with 1 star reviews versus those associated with 5 star reviews. These differences may not be just related to sentiment. We might want to find words used more often by Democratic than Republican

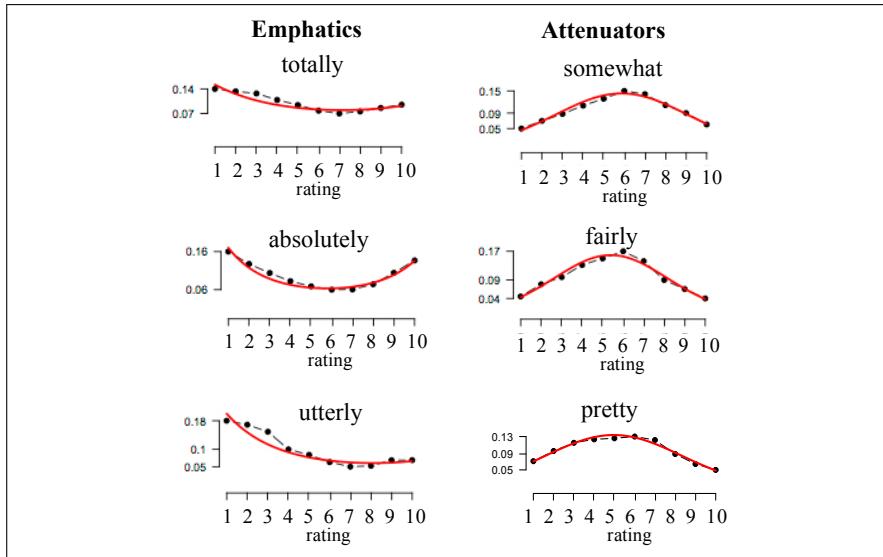


Figure 21.11 Potts diagrams (Potts, 2011) for emphatic and attenuating adverbs.

members of Congress, or words used more often in menus of expensive restaurants than cheap restaurants.

Given two classes of documents, to find words more associated with one category than another, we might choose to just compute the difference in frequencies (is a word w more frequent in class A or class B ?). Or instead of the difference in frequencies we might want to compute the ratio of frequencies, or the log odds ratio (the log of the ratio between the odds of the two words). Then we can sort words by whichever of these associations with the category we use, (sorting from words overrepresented in category A to words overrepresented in category B).

The problem with simple log-likelihood or log odds methods is that they don't work well for very rare words or very frequent words; for words that are very frequent, all differences seem large, and for words that are very rare, no differences seem large.

In this section we walk through the details of one solution to this problem: the "log odds ratio informative Dirichlet prior" method of Monroe et al. (2008) that is a particularly useful method for finding words that are statistically overrepresented in one particular category of texts compared to another. It's based on the idea of using another large corpus to get a prior estimate of what we expect the frequency of each word to be.

Let's start with the goal: assume we want to know whether the word *horrible* occurs more in corpus i or corpus j . We could compute the **log likelihood ratio**, using $f^i(w)$ to mean the frequency of word w in corpus i , and n^i to mean the total number of words in corpus i :

$$\begin{aligned} \text{lhr}(\text{horrible}) &= \log \frac{P^i(\text{horrible})}{P^j(\text{horrible})} \\ &= \log P^i(\text{horrible}) - \log P^j(\text{horrible}) \\ &= \log \frac{f^i(\text{horrible})}{n^i} - \log \frac{f^j(\text{horrible})}{n^j} \end{aligned} \tag{21.7}$$

log odds ratio Instead, let's compute the **log odds ratio**: does *horrible* have higher odds in i or in

j:

$$\begin{aligned}
 \text{lor}(\text{horrible}) &= \log \left(\frac{P^i(\text{horrible})}{1 - P^i(\text{horrible})} \right) - \log \left(\frac{P^j(\text{horrible})}{1 - P^j(\text{horrible})} \right) \\
 &= \log \left(\frac{\frac{f^i(\text{horrible})}{n^i}}{1 - \frac{f^i(\text{horrible})}{n^i}} \right) - \log \left(\frac{\frac{f^j(\text{horrible})}{n^j}}{1 - \frac{f^j(\text{horrible})}{n^j}} \right) \\
 &= \log \left(\frac{f^i(\text{horrible})}{n^i - f^i(\text{horrible})} \right) - \log \left(\frac{f^j(\text{horrible})}{n^j - f^j(\text{horrible})} \right) \quad (21.8)
 \end{aligned}$$

The Dirichlet intuition is to use a large background corpus to get a prior estimate of what we expect the frequency of each word w to be. We'll do this very simply by adding the counts from that corpus to the numerator and denominator, so that we're essentially shrinking the counts toward that prior. It's like asking how large are the differences between i and j given what we would expect given their frequencies in a well-estimated large background corpus.

The method estimates the difference between the frequency of word w in two corpora i and j via the prior-modified log odds ratio for w , $\delta_w^{(i-j)}$, which is estimated as:

$$\delta_w^{(i-j)} = \log \left(\frac{f_w^i + \alpha_w}{n^i + \alpha_0 - (f_w^i + \alpha_w)} \right) - \log \left(\frac{f_w^j + \alpha_w}{n^j + \alpha_0 - (f_w^j + \alpha_w)} \right) \quad (21.9)$$

(where n^i is the size of corpus i , n^j is the size of corpus j , f_w^i is the count of word w in corpus i , f_w^j is the count of word w in corpus j , α_0 is the size of the background corpus, and α_w is the count of word w in the background corpus.)

In addition, Monroe et al. (2008) make use of an estimate for the variance of the log-odds-ratio:

$$\sigma^2 \left(\hat{\delta}_w^{(i-j)} \right) \approx \frac{1}{f_w^i + \alpha_w} + \frac{1}{f_w^j + \alpha_w} \quad (21.10)$$

The final statistic for a word is then the z-score of its log-odds-ratio:

$$\frac{\hat{\delta}_w^{(i-j)}}{\sqrt{\sigma^2 \left(\hat{\delta}_w^{(i-j)} \right)}} \quad (21.11)$$

The Monroe et al. (2008) method thus modifies the commonly used log odds ratio in two ways: it uses the z-scores of the log odds ratio, which controls for the amount of variance in a word's frequency, and it uses counts from a background corpus to provide a prior count for words.

Fig. 21.12 shows the method applied to a dataset of restaurant reviews from Yelp, comparing the words used in 1-star reviews to the words used in 5-star reviews (Jurafsky et al., 2014). The largest difference is in obvious sentiment words, with the 1-star reviews using negative sentiment words like *worse*, *bad*, *awful* and the 5-star reviews using positive sentiment words like *great*, *best*, *amazing*. But there are other illuminating differences. 1-star reviews use logical negation (*no*, *not*), while 5-star reviews use emphatics and emphasize universality (*very*, *highly*, *every*, *always*). 1-star reviews use first person plurals (*we*, *us*, *our*) while 5 star reviews use the second person. 1-star reviews talk about people (*manager*, *waiter*, *customer*) while 5-star reviews talk about dessert and properties of expensive restaurants like courses and atmosphere. See Jurafsky et al. (2014) for more details.

Class	Words in 1-star reviews	Class	Words in 5-star reviews
Negative	worst, rude, terrible, horrible, bad, awful, disgusting, bland, tasteless, gross, mediocre, overpriced, worse, poor	Positive	great, best, love(d), delicious, amazing, favorite, perfect, excellent, awesome, friendly, fantastic, fresh, wonderful, incredible, sweet, yum(my)
Negation	no, not	Emphatics/universals	very, highly, perfectly, definitely, absolutely, everything, every, always
1Pl pro	we, us, our	2 pro	you
3 pro	she, he, her, him	Articles	a, the
Past verb	was, were, asked, told, said, did, charged, waited, left, took	Advice	try, recommend
Sequencers	after, then	Conjunct	also, as, well, with, and
Nouns	manager, waitress, waiter, customer, customers, attitude, waste, poisoning, money, bill, minutes	Nouns	atmosphere, dessert, chocolate, wine, course, menu
Irrealis modals	would, should	Auxiliaries	is/'s, can, 've, are
Comp	to, that	Prep, other	in, of, die, city, mouth

Figure 21.12 The top 50 words associated with one-star and five-star restaurant reviews in a Yelp dataset of 900,000 reviews, using the Monroe et al. (2008) method (Jurafsky et al., 2014).

21.6 Using Lexicons for Sentiment Recognition

In Chapter 4 we introduced the naive Bayes algorithm for sentiment analysis. The lexicons we have focused on throughout the chapter so far can be used in a number of ways to improve sentiment detection.

In the simplest case, lexicons can be used when we don't have sufficient training data to build a supervised sentiment analyzer; it can often be expensive to have a human assign sentiment to each document to train the supervised classifier.

In such situations, lexicons can be used in a rule-based algorithm for classification. The simplest version is just to use the ratio of positive to negative words: if a document has more positive than negative words (using the lexicon to decide the polarity of each word in the document), it is classified as positive. Often a threshold λ is used, in which a document is classified as positive only if the ratio is greater than λ . If the sentiment lexicon includes positive and negative weights for each word, θ_w^+ and θ_w^- , these can be used as well. Here's a simple such sentiment algorithm:

$$\begin{aligned}
 f^+ &= \sum_{w \text{ s.t. } w \in \text{positivelexicon}} \theta_w^+ \text{count}(w) \\
 f^- &= \sum_{w \text{ s.t. } w \in \text{negativelexicon}} \theta_w^- \text{count}(w) \\
 \text{sentiment} &= \begin{cases} + & \text{if } \frac{f^+}{f^-} > \lambda \\ - & \text{if } \frac{f^-}{f^+} > \lambda \\ 0 & \text{otherwise.} \end{cases} \tag{21.12}
 \end{aligned}$$

If supervised training data is available, these counts computed from sentiment lexicons, sometimes weighted or normalized in various ways, can also be used as features in a classifier along with other lexical or non-lexical features. We return to such algorithms in Section 21.8.

21.7 Other tasks: Personality

personality Many other kinds of affective meaning can be extracted from text and speech. For example detecting a person's **personality** from their language can be useful for dialog systems (users tend to prefer agents that match their personality), and can play a useful role in computational social science questions like understanding how personality is related to other kinds of behavior.

Many theories of human personality are based around a small number of dimensions, such as various versions of the "Big Five" dimensions ([Digman, 1990](#)):

- Extroversion vs. Introversion:** sociable, assertive, playful vs. aloof, reserved, shy
- Emotional stability vs. Neuroticism:** calm, unemotional vs. insecure, anxious
- Agreeableness vs. Disagreeableness:** friendly, cooperative vs. antagonistic, fault-finding
- Conscientiousness vs. Unconscientiousness:** self-disciplined, organized vs. inefficient, careless
- Openness to experience:** intellectual, insightful vs. shallow, unimaginative

A few corpora of text and speech have been labeled for the personality of their author by having the authors take a standard personality test. The essay corpus of [Pennebaker and King \(1999\)](#) consists of 2,479 essays (1.9 million words) from psychology students who were asked to "write whatever comes into your mind" for 20 minutes. The EAR (Electronically Activated Recorder) corpus of [Mehl et al. \(2006\)](#) was created by having volunteers wear a recorder throughout the day, which randomly recorded short snippets of conversation throughout the day, which were then transcribed. The Facebook corpus of [\(Schwartz et al., 2013\)](#) includes 309 million words of Facebook posts from 75,000 volunteers.

For example, here are samples from [Pennebaker and King \(1999\)](#) from an essay written by someone on the neurotic end of the neurotic/emotionally stable scale,

One of my friends just barged in, and I jumped in my seat. This is crazy.
I should tell him not to do that again. I'm not that fastidious actually.
But certain things annoy me. The things that would annoy me would
actually annoy any normal human being, so I know I'm not a freak.

and someone on the emotionally stable end of the scale:

I should excel in this sport because I know how to push my body harder
than anyone I know, no matter what the test I always push my body
harder than everyone else. I want to be the best no matter what the sport
or event. I should also be good at this because I love to ride my bike.

interpersonal stance Another kind of affective meaning is what [Scherer \(2000\)](#) calls **interpersonal stance**, the 'affective stance taken toward another person in a specific interaction coloring the interpersonal exchange'. Extracting this kind of meaning means automatically labeling participants for whether they are friendly, supportive, distant. For example [Ranganath et al. \(2013\)](#) studied a corpus of speed-dates, in which participants went on a series of 4-minute romantic dates, wearing microphones. Each participant labeled each other for how flirtatious, friendly, awkward, or assertive they were. [Ranganath et al. \(2013\)](#) then used a combination of lexicons and other features to detect these interpersonal stances from text.

21.8 Affect Recognition

Detection of emotion, personality, interactional stance, and the other kinds of affective meaning described by [Scherer \(2000\)](#) can be done by generalizing the algorithms described above for detecting sentiment.

The most common algorithms involve supervised classification: a training set is labeled for the affective meaning to be detected, and a classifier is built using features extracted from the training set. As with sentiment analysis, if the training set is large enough, and the test set is sufficiently similar to the training set, simply using all the words or all the bigrams as features in a powerful classifier like SVM or logistic regression, as described in Fig. 4.2 in Chapter 4, is an excellent algorithm whose performance is hard to beat. Thus we can treat affective meaning classification of a text sample as simple document classification.

Some modifications are nonetheless often necessary for very large datasets. For example, the [Schwartz et al. \(2013\)](#) study of personality, gender, and age using 700 million words of Facebook posts used only a subset of the n-grams of lengths 1-3. Only words and phrases used by at least 1% of the subjects were included as features, and 2-grams and 3-grams were only kept if they had sufficiently high PMI (PMI greater than $2 * \text{length}$, where length is the number of words):

$$\text{pmi}(\text{phrase}) = \log \frac{p(\text{phrase})}{\prod_{w \in \text{phrase}} p(w)} \quad (21.13)$$

Various weights can be used for the features, including the raw count in the training set, or some normalized probability or log probability. [Schwartz et al. \(2013\)](#), for example, turn feature counts into phrase likelihoods by normalizing them by each subject's total word use.

$$p(\text{phrase} | \text{subject}) = \frac{\text{freq}(\text{phrase}, \text{subject})}{\sum_{\text{phrase}' \in \text{vocab}(\text{subject})} \text{freq}(\text{phrase}', \text{subject})} \quad (21.14)$$

If the training data is sparser, or not as similar to the test set, any of the lexicons we've discussed can play a helpful role, either alone or in combination with all the words and n-grams.

Many possible values can be used for lexicon features. The simplest is just an indicator function, in which the value of a feature f_L takes the value 1 if a particular text has any word from the relevant lexicon L . Using the notation of Chapter 4, in which a feature value is defined for a particular output class c and document x .

$$f_L(c, x) = \begin{cases} 1 & \text{if } \exists w : w \in L \ \& \ w \in x \ \& \ \text{class} = c \\ 0 & \text{otherwise} \end{cases}$$

Alternatively the value of a feature f_L for a particular lexicon L can be the total number of word *tokens* in the document that occur in L :

$$f_L = \sum_{w \in L} \text{count}(w)$$

For lexica in which each word is associated with a score or weight, the count can be multiplied by a weight θ_w^L :

$$f_L = \sum_{w \in L} \theta_w^L \text{count}(w)$$

Counts can alternatively be logged or normalized per writer as in Eq. 21.14.

However they are defined, these lexicon features are then used in a supervised classifier to predict the desired affective category for the text or document. Once a classifier is trained, we can examine which lexicon features are associated with which classes. For a classifier like logistic regression the feature weight gives an indication of how associated the feature is with the class.

Thus, for example, Mairesse and Walker (2008) found that for classifying personality, for the dimension *Agreeable*, the LIWC lexicons *Family* and *Home* were positively associated while the LIWC lexicons *anger* and *swear* were negatively associated. By contrast, Extroversion was positively associated with the *Friend*, *Religion* and *Self* lexicons, and Emotional Stability was positively associated with *Sports* and negatively associated with *Negative Emotion*.

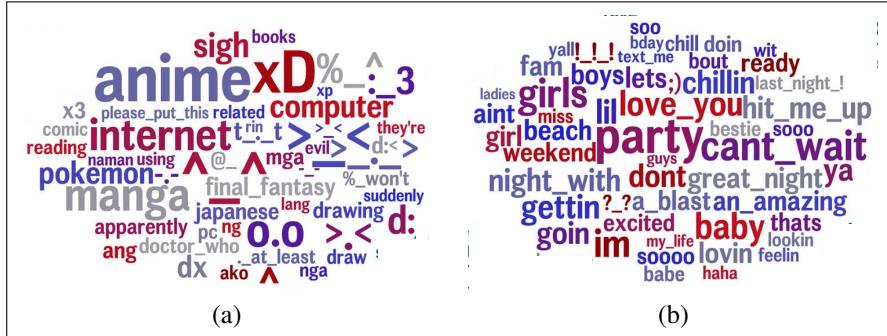


Figure 21.13 Word clouds from Schwartz et al. (2013), showing words highly associated with introversion (left) or extroversion (right). The size of the word represents the association strength (the regression coefficient), while the color (ranging from cold to hot) represents the relative frequency of the word/phrase (from low to high).

In the situation in which we use all the words and phrases in the document as potential features, we can use the resulting weights from the learned regression classifier as the basis of an affective lexicon. In the Extroversion/Introversion classifier of Schwartz et al. (2013), ordinary least-squares regression is used to predict the value of a personality dimension from all the words and phrases. The resulting regression coefficient for each word or phrase can be used as an association value with the predicted dimension. The word clouds in Fig. 21.13 show an example of words associated with introversion (a) and extroversion (b).

21.9 Lexicon-based methods for Entity-Centric Affect

What if we want to get an affect score not for an entire document, but for a particular entity in the text? The entity-centric method of Field and Tsvetkov (2019) combines affect lexicons with contextual embeddings to assign an affect score to an entity in text. In the context of affect about people, they relabel the Valence/Arousal/Dominance dimension as Sentiment/Agency/Power. The algorithm first trains classifiers to map embeddings to scores:

1. For each word w in the training corpus:
 - (a) Use off-the-shelf pre-trained language models (ELMo or BERT) to extract a contextual embedding e for each instance of the word. No addi-

tional fine-tuning is done.

- (b) Average over the \mathbf{e} embeddings of each instance of w to obtain a single embedding vector for one training point w .
 - (c) Use the NRC VAD Lexicon to get S, A, and P scores for w .
2. Train (three) regression models on all words w to predict V, A, D scores from a word's average embedding.

Now given an entity mention m in a text, we assign affect scores as follows:

1. Use the same pre-trained LM to get contextual embeddings for m in context.
2. Feed this embeddings through the 3 regression models to get S, A, P scores for the entity.

This results in a (S,A,P) tuple for a given entity mention; it get scores for the representation of an entity in a complete document, we can run coref and average the (S,A,P) scores for all the mentions. They find ELMo works much better than BERT. Fig. 21.14 shows the scores from their algorithm for characters from the movie *The Dark Knight* when run on Wikipedia plot summary texts with gold coreference.



Figure 21.14 Power (dominance), sentiment (valence) and agency (arousal) for characters in the movie *The Dark Knight* computed from ELMo embeddings trained on the NRC VAD Lexicon. Note the protagonist (Batman) and the antagonist (the Joker) have high power and agency scores but differ in sentiment, while the love interest Rachel has low power and agency but high sentiment.

21.10 Connotation Frames

connotation frame

The lexicons we've described so far define a word as a point in affective space. A **connotation frame**, by contrast, is a lexicon that incorporates a richer kind of grammatical structure, by combining affective lexicons with the frame semantic lexicons of Chapter 20. The basic insight of connotation frame lexicons is that a predicate like a verb expresses connotations about the verb's arguments (Rashkin et al. 2016, Rashkin et al. 2017).

Consider sentences like:

- (21.15) Country A violated the sovereignty of Country B
 (21.16) the teenager ... survived the Boston Marathon bombing”

By using the verb *violate* in (21.15), the author is expressing their sympathies with Country B, portraying Country B as a victim, and expressing antagonism toward the agent Country A. By contrast, in using the verb *survive*, the author of (21.16) is expressing that the bombing is a negative experience, and the subject of the sentence, the teenager, is a sympathetic character. These aspects of connotation are inherent in the meaning of the verbs *violate* and *survive*, as shown in Fig. 21.15.

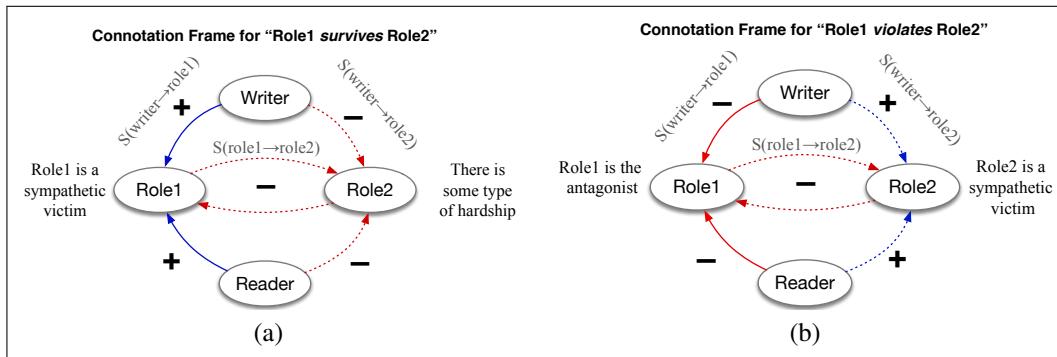


Figure 21.15 Connotation frames for *survive* and *violate*. (a) For *survive*, the writer and reader have positive sentiment toward Role1, the subject, and negative sentiment toward Role2, the direct object. (b) For *violate*, the writer and reader have positive sentiment instead toward Role2, the direct object.

The connotation frame lexicons of Rashkin et al. (2016) and Rashkin et al. (2017) also express other connotative aspects of the predicate toward each argument, including the *effect* (something bad happened to x) *value*: (x is valuable), and *mental state*: (x is distressed by the event). Connotation frames can also mark aspects of power and agency; see Chapter 20 (Sap et al., 2017).

Connotation frames can be built by hand (Sap et al., 2017), or they can be learned by supervised learning (Rashkin et al., 2016), for example using hand-labeled training data to supervise classifiers for each of the individual relations, e.g., whether $S(\text{writer} \rightarrow \text{Role1})$ is + or -, and then improving accuracy via global constraints across all relations.

21.11 Summary

- Many kinds of affective states can be distinguished, including *emotions*, *moods*, *attitudes* (which include *sentiment*), *interpersonal stance*, and *personality*.
- **Emotion** can be represented by fixed atomic units often called **basic emotions**, or as points in space defined by dimensions like **valence** and **arousal**.
- Words have **connotational** aspects related to these affective states, and this connotational aspect of word meaning can be represented in lexicons.
- Affective lexicons can be built by hand, using **crowd sourcing** to label the affective content of each word.
- Lexicons can be built with **semi-supervised**, bootstrapping from seed words using similarity metrics like embedding cosine.
- Lexicons can be learned in a **fully supervised** manner, when a convenient training signal can be found in the world, such as ratings assigned by users on a review site.

- Words can be assigned weights in a lexicon by using various functions of word counts in training texts, and ratio metrics like **log odds ratio informative Dirichlet prior**.
- Personality is often represented as a point in 5-dimensional space.
- Affect can be detected, just like sentiment, by using standard supervised **text classification** techniques, using all the words or bigrams in a text as features. Additional features can be drawn from counts of words in lexicons.
- Lexicons can also be used to detect affect in a **rule-based classifier** by picking the simple majority sentiment based on counts of words in each lexicon.
- **Connotation frames** express richer relations of affective meaning that a predicate encodes about its arguments.

Bibliographical and Historical Notes

The idea of formally representing the subjective meaning of words began with [Osgood et al. \(1957\)](#), the same pioneering study that first proposed the vector space model of meaning described in Chapter 6. [Osgood et al. \(1957\)](#) had participants rate words on various scales, and ran factor analysis on the ratings. The most significant factor they uncovered was the evaluative dimension, which distinguished between pairs like *good/bad*, *valuable/worthless*, *pleasant/unpleasant*. This work influenced the development of early dictionaries of sentiment and affective meaning in the field of **content analysis** ([Stone et al., 1966](#)).

subjectivity

[Wiebe \(1994\)](#) began an influential line of work on detecting **subjectivity** in text, beginning with the task of identifying subjective sentences and the subjective characters who are described in the text as holding private states, beliefs or attitudes. Learned sentiment lexicons such as the polarity lexicons of [Hatzivassiloglou and McKeown \(1997\)](#) were shown to be a useful feature in subjectivity detection ([Hatzivassiloglou and Wiebe 2000, Wiebe 2000](#)).

The term **sentiment** seems to have been introduced in 2001 by [Das and Chen \(2001\)](#), to describe the task of measuring market sentiment by looking at the words in stock trading message boards. In the same paper [Das and Chen \(2001\)](#) also proposed the use of a sentiment lexicon. The list of words in the lexicon was created by hand, but each word was assigned weights according to how much it discriminated a particular class (say buy versus sell) by maximizing across-class variation and minimizing within-class variation. The term *sentiment*, and the use of lexicons, caught on quite quickly (e.g., *inter alia*, [Turney 2002](#)). [Pang et al. \(2002\)](#) first showed the power of using all the words without a sentiment lexicon; see also [Wang and Manning \(2012\)](#).

Most of the semi-supervised methods we describe for extending sentiment dictionaries drew on the early idea that synonyms and antonyms tend to co-occur in the same sentence. ([Miller and Charles 1991, Justeson and Katz 1991, Riloff and Shepherd 1997](#)). Other semi-supervised methods for learning cues to affective meaning rely on information extraction techniques, like the AutoSlog pattern extractors ([Riloff and Wiebe, 2003](#)). Graph based algorithms for sentiment were first suggested by [Hatzivassiloglou and McKeown \(1997\)](#), and graph propagation became a standard method ([Zhu and Ghahramani 2002, Zhu et al. 2003, Zhou et al. 2004, Velikovich et al. 2010](#)). Crowdsourcing can also be used to improve precision by

filtering the result of semi-supervised lexicon learning ([Riloff and Shepherd 1997](#), [Fast et al. 2016](#)).

Much recent work focuses on ways to learn embeddings that directly encode sentiment or other properties, such as the DENSIFIER algorithm of [Rothe et al. \(2016\)](#) that learns to transform the embedding space to focus on sentiment (or other) information.

CHAPTER

22

Coreference Resolution

and even Stigand, the patriotic archbishop of Canterbury, found it advisable—”

‘Found WHAT?’ said the Duck.

‘Found IT,’ the Mouse replied rather crossly: ‘of course you know what “it” means.’

‘I know what “it”means well enough, when I find a thing,’ said the Duck: ‘it’s generally a frog or a worm. The question is, what did the archbishop find?’

Lewis Carroll, *Alice in Wonderland*

An important component of language understanding is knowing *who* is being talked about in a text. Consider the following passage:

- (22.1) Victoria Chen, CFO of Megabucks Banking, saw her pay jump to \$2.3 million, as the 38-year-old became the company’s president. It is widely known that she came to Megabucks from rival Lotsabucks.

Each of the underlined phrases in this passage is used by the writer to refer to a person named Victoria Chen. We call linguistic expressions like *her* or *Victoria Chen* **mentions** or **referring expressions**, and the discourse entity that is referred to (Victoria Chen) the **referent**. (To distinguish between referring expressions and their referents, we italicize the former.)¹ Two or more referring expressions that are used to refer to the same discourse entity are said to **corefer**; thus, *Victoria Chen* and *she* corefer in (22.1).

Coreference is an important component of natural language understanding. A dialogue system that has just told the user “*There is a 2pm flight on United and a 4pm one on Cathay Pacific*” must know which flight the user means by “*I’ll take the Cathay Pacific flight*”. A question answering system that uses Wikipedia to answer a question about where Marie Curie was born must know who *she* was in the sentence “*She was born in Warsaw*”. And a machine translation system translating from a language like Spanish, in which pronouns can be dropped, must use coreference from the previous sentence to decide whether the Spanish sentence “*Me incanta el conocimiento*”, *dice*.’ should be translated as “*I love knowledge*”, *he said*’, or “*I love knowledge*”, *she said*’. Indeed, this example comes from an actual news article about a female professor and was mistranslated as “he” by Google Translate because of inaccurate coreference resolution (Schiebinger, 2019).

Natural language understanding systems (and humans) interpret linguistic expressions with respect to a **discourse model** (Karttunen, 1969) shown in Fig. 22.1. A discourse model is a mental model that the system (or a human hearer) builds incrementally as it interprets a text, containing representations of the entities referred to in the text, as well as properties of the entities and relations among them. When a referent is first mentioned in a discourse, we say that a representation for it is **evoked** into the model. Upon subsequent mention, this representation is **accessed** from the

mention

referent

corefer

discourse model

evoked
accessed

¹ As a convenient shorthand, we sometimes speak of a referring expression referring to a referent, e.g., saying that *she* refers to Victoria Chen. However, the reader should keep in mind that what we really mean is that the speaker is performing the act of referring to Victoria Chen by uttering *she*.

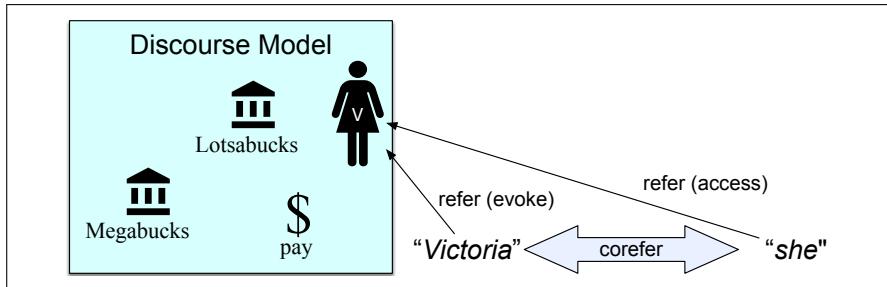


Figure 22.1 How mentions evoke and access discourse entities in a discourse model.

model.

Reference in a text to an entity that has been previously introduced into the discourse is called **anaphora**, and the referring expression used is said to be an **anaphor**, or anaphoric.² In passage (22.1), the pronouns *she* and *her* and the definite NP *the 38-year-old* are therefore anaphoric. The anaphor corefers with a prior mention (in this case *Victoria Chen*) that is called the **antecedent**. Not every referring expression is an antecedent. An entity that has only a single mention in a text (like *Lotsabucks* in (22.1)) is called a **singleton**.

In this chapter we focus on the task of **coreference resolution**. Coreference resolution is the task of determining whether two mentions *corefer*, by which we mean they refer to the same entity in the discourse model (the same *discourse entity*). The set of corefering expressions is often called a **coreference chain** or a **cluster**. For example, in processing (22.1), a coreference resolution algorithm would need to find at least four coreference chains, corresponding to the four entities in the discourse model in Fig. 22.1.

1. *{Victoria Chen, her, the 38-year-old, She}*
2. *{Megabucks Banking, the company, Megabucks}*
3. *{her pay}*
4. *{Lotsabucks}*

Note that mentions can be nested; for example the mention *her* is syntactically part of another mention, *her pay*, referring to a completely different discourse entity.

Coreference resolution thus comprises two tasks (although they are often performed jointly): (1) identifying the mentions, and (2) clustering them into coreference chains/discourse entities.

We said that two mentions *corefered* if they are associated with the same *discourse entity*. But often we'd like to go further, deciding which real world entity is associated with this discourse entity. For example, the mention *Washington* might refer to the US state, or the capital city, or the person George Washington; the interpretation of the sentence will of course be very different for these completely different named entity types (Chapter 18). The task of **entity linking** (Ji and Grishman, 2011) or *entity resolution* is the task of mapping a discourse entity to some real-world individual.³ We usually operationalize entity linking or resolution by

² We will follow the common NLP usage of *anaphor* to mean any mention that has an antecedent, rather than the more narrow usage to mean only mentions (like pronouns) whose interpretation depends on the antecedent (under the narrower interpretation, repeated names are not anaphors).

³ Computational linguistics/NLP thus differs in its use of the term *reference* from the field of formal semantics, which uses the words *reference* and *coreference* to describe the relation between a mention and a real-world entity. By contrast, we follow the functional linguistics tradition in which a mention *refers* to a *discourse entity* (Webber, 1978) and the relation between a discourse entity and the real world

mapping to an *ontology*: a list of entities in the world, like a gazeteer (Chapter 16). Perhaps the most common ontology used for this task is Wikipedia; each Wikipedia page acts as the unique id for a particular entity. Thus the entity linking task of **wikification** (Mihalcea and Csomai, 2007) is the task of deciding which Wikipedia page corresponding to an individual is being referred to by a mention. But entity linking can be done with any ontology; for example if we have an ontology of genes, we can link mentions of genes in text to the disambiguated gene name in the ontology.

In the next sections we introduce the task of coreference resolution in more detail, and offer a variety of architectures for resolution, from simple deterministic baseline algorithms to state-of-the-art neural models.

Before turning to algorithms, however, we mention some important tasks we will only touch on briefly at the end of this chapter. First are the famous Winograd Schema problems (so-called because they were first pointed out by Terry Winograd in his dissertation). These entity coreference resolution problems are designed to be too difficult to be solved by the resolution methods we describe in this chapter, and the kind of real-world knowledge they require has made them a kind of challenge task for natural language understanding. For example, consider the task of determining the correct antecedent of the pronoun *they* in the following example:

- (22.2) The city council denied the demonstrators a permit because

- a. they feared violence.
- b. they advocated violence.

Determining the correct antecedent for the pronoun *they* requires understanding that the second clause is intended as an explanation of the first clause, and also that city councils are perhaps more likely than demonstrators to fear violence and that demonstrators might be more likely to advocate violence. Solving Winograd Schema problems requires finding way to represent or discover the necessary real world knowledge.

A problem we won't discuss in this chapter is the related task of **event coreference**, deciding whether two event mentions (such as the *buy* and the *acquisition* in these two sentences from the ECB+ corpus) refer to the same event:

- (22.3) AMD agreed to [**buy**] Markham, Ontario-based ATI for around \$5.4 billion in cash and stock, the companies announced Monday.
 (22.4) The [**acquisition**] would turn AMD into one of the world's largest providers of graphics chips.

Event mentions are much harder to detect than entity mentions, since they can be verbal as well as nominal. Once detected, the same mention-pair and mention-ranking models used for entities are often applied to events.

An even more complex kind of coreference is **discourse deixis** (Webber, 1988), in which an anaphor refers back to a discourse segment, which can be quite hard to delimit or categorize, like the examples in (22.5) adapted from Webber (1991):

- (22.5) According to Soleil, Beau just opened a restaurant
- a. But *that* turned out to be a lie.
 - b. But *that* was false.
 - c. *That* struck me as a funny way to describe the situation.

The referent of *that* is a speech act (see Chapter 26) in (22.5a), a proposition in (22.5b), and a manner of description in (22.5c). The field awaits the development of robust methods for interpreting most of these types of reference.

individual requires an additional step of *linking*.

event coreference

discourse deixis

22.1 Coreference Phenomena: Linguistic Background

We now offer some linguistic background on reference phenomena. We introduce the four types of referring expressions (definite and indefinite NPs, pronouns, and names), describe how these are used to evoke and access entities in the discourse model, and talk about linguistic features of the anaphor/antecedent relation (like number/gender agreement, or properties of verb semantics).

22.1.1 Types of Referring Expressions

Indefinite Noun Phrases: The most common form of indefinite reference in English is marked with the determiner *a* (or *an*), but it can also be marked by a quantifier such as *some* or even the determiner *this*. Indefinite reference generally introduces into the discourse context entities that are new to the hearer.

- (22.6) a. Mrs. Martin was so very kind as to send Mrs. Goddard *a beautiful goose*.
 b. He had gone round one day to bring her *some walnuts*.
 c. I saw *this beautiful cauliflower* today.

Definite Noun Phrases: Definite reference, such as via NPs that use the English article *the*, refers to an entity that is identifiable to the hearer. An entity can be identifiable to the hearer because it has been mentioned previously in the text and thus is already represented in the discourse model:

- (22.7) It concerns a white stallion which I have sold to an officer. But the pedigree of *the white stallion* was not fully established.

Alternatively, an entity can be identifiable because it is contained in the hearer's set of beliefs about the world, or the uniqueness of the object is implied by the description itself, in which case it evokes a representation of the referent into the discourse model, as in (22.9):

- (22.8) I read about it in the *New York Times*.
- (22.9) Have you seen the car keys?

These last uses are quite common; more than half of definite NPs in newswire texts are non-anaphoric, often because they are the first time an entity is mentioned (Poesio and Vieira 1998, Bean and Riloff 1999).

Pronouns: Another form of definite reference is pronominalization, used for entities that are extremely salient in the discourse, (as we discuss below):

- (22.10) Emma smiled and chatted as cheerfully as *she* could,

Pronouns can also participate in **cataphora**, in which they are mentioned before their referents are, as in (22.11).

- (22.11) Even before *she* saw *it*, Dorothy had been thinking about the Emerald City every day.

Here, the pronouns *she* and *it* both occur *before* their referents are introduced.

Pronouns also appear in quantified contexts in which they are considered to be **bound**, as in (22.12).

- (22.12) Every dancer brought *her* left arm forward.

Under the relevant reading, *her* does not refer to some woman in context, but instead behaves like a variable bound to the quantified expression *every dancer*. We are not concerned with the bound interpretation of pronouns in this chapter.

cataphora

bound

In some languages, pronouns can appear as clitics attached to a word, like *lo* ('it') in this Spanish example from AnCora (Recasens and Martí, 2010):

- (22.13) La intención es reconocer el gran prestigio que tiene la maratón y unirlo con esta gran carrera.
 'The aim is to recognize the great prestige that the Marathon has and join|it with this great race.'

Demonstrative Pronouns: Demonstrative pronouns *this* and *that* can appear either alone or as determiners, for instance, *this ingredient*, *that spice*:

- (22.14) I just bought a copy of Thoreau's *Walden*. I had bought one five years ago.
That one had been very tattered; *this one* was in much better condition.

Note that *this NP* is ambiguous; in colloquial spoken English, it can be indefinite, as in (22.6), or definite, as in (22.14).

zero anaphor

Zero Anaphora: Instead of using a pronoun, in some languages (including Chinese, Japanese, and Italian) it is possible to have an anaphor that has no lexical realization at all, called a **zero anaphor** or zero pronoun, as in the following Italian and Japanese examples from Poesio et al. (2016):

- (22.15) EN [John]_i went to visit some friends. On the way [he]_i bought some wine.
 IT [Giovanni]_i andò a far visita a degli amici. Per via ϕ_i comprò del vino.
 JA [John]_i-wa yujin-o houmon-sita. Tochu-de ϕ_i wain-o ka-tta.

or this Chinese example:

- (22.16) [我] 前一会精神上太紧张。[0] 现在比较平静了
 [I] was too nervous a while ago. ... [0] am now calmer.

Zero anaphors complicate the task of mention detection in these languages.

Names: Names (such as of people, locations, or organizations) can be used to refer to both new and old entities in the discourse:

- (22.17) a. **Miss Woodhouse** certainly had not done him justice.
 b. **International Business Machines** sought patent compensation from Amazon; **IBM** had previously sued other companies.

information status discourse-new discourse-old

22.1.2 Information Status

The way referring expressions are used to evoke new referents into the discourse (introducing new information), or access old entities from the model (old information), is called their **information status** or **information structure**. Entities can be **discourse-new** or **discourse-old**, and indeed it is common to distinguish at least three kinds of entities informationally (Prince, 1981a):

new NPs:

brand new NPs: these introduce entities that are discourse-new and hearer-new like *a fruit* or *some walnuts*.

unused NPs: these introduce entities that are discourse-new but hearer-old (like *Hong Kong*, *Marie Curie*, or *the New York Times*).

old NPs: also called **evoked NPs**, these introduce entities that already in the discourse model, hence are both discourse-old and hearer-old, like *it* in "*I went to a new restaurant. It was...*".

inferredables: these introduce entities that are neither hearer-old nor discourse-old, but the hearer can infer their existence by reasoning based on other entities that are in the discourse. Consider the following examples:

(22.18) I went to a superb restaurant yesterday. *The chef* had just opened it.

(22.19) Mix flour, butter and water. Knead *the dough* until shiny.

bridging inference

Neither *the chef* nor *the dough* were in the discourse model based on the first sentence of either example, but the reader can make a **bridging inference** that these entities should be added to the discourse model and associated with the restaurant and the ingredients, based on world knowledge that restaurants have chefs and dough is the result of mixing flour and liquid (Haviland and Clark 1974, Webber and Baldwin 1992, Nissim et al. 2004, Hou et al. 2018).

given-new

The form of an NP gives strong clues to its information status. We often talk about an entity's position on the **given-new** dimension, the extent to which the referent is **given** (salient in the discourse, easier for the hearer to call to mind, predictable by the hearer), versus **new** (non-salient in the discourse, unpredictable) (Chafe 1976, Prince 1981b, Gundel et al. 1993). A referent that is very **accessible** (Ariel, 2001) i.e., very salient in the hearer's mind or easy to call to mind, can be referred to with less linguistic material. For example pronouns are used only when the referent has a high degree of activation or **salience** in the discourse model.⁴ By contrast, less salient entities, like a new referent being introduced to the discourse, will need to be introduced with a longer and more explicit referring expression to help the hearer recover the referent.

accessible

Thus when an entity is first introduced into a discourse its mentions are likely to have full names, titles or roles, or appositive or restrictive relative clauses, as in the introduction of our protagonist in (22.1): *Victoria Chen, CFO of Megabucks Banking*. As an entity is discussed over a discourse, it becomes more salient to the hearer and its mentions on average typically becomes shorter and less informative, for example with a shortened name (for example *Ms. Chen*), a definite description (*the 38-year-old*), or a pronoun (*she* or *her*) (Hawkins 1978). However, this change in length is not monotonic, and is sensitive to discourse structure (Grosz 1977b, Reichman 1985, Fox 1993).

salience

22.1.3 Complications: Non-Referring Expressions

Many noun phrases or other nominals are not referring expressions, although they may bear a confusing superficial resemblance. For example in some of the earliest computational work on reference resolution, Karttunen (1969) pointed out that the NP *a car* in the following example does not create a discourse referent:

(22.20) Janet doesn't have *a car*.

and cannot be referred back to by anaphoric *it* or *the car*:

(22.21) **It* is a Toyota.

(22.22) **The car* is red.

We summarize here four common types of structures that are not counted as mentions in coreference tasks and hence complicate the task of mention-detection:

⁴ Pronouns also usually (but not always) refer to entities that were introduced no further than one or two sentences back in the ongoing discourse, whereas definite noun phrases can often refer further back.

Appositives: An appositional structure is a noun phrase that appears next to a head noun phrase, describing the head. In English they often appear in commas, like “a unit of UAL” appearing in apposition to the NP *United*, or *CFO of Megabucks Banking* in apposition to *Victoria Chen*.

- (22.23) Victoria Chen, CFO of Megabucks Banking, saw ...
 (22.24) United, a unit of UAL, matched the fares.

Appositional NPs are not referring expressions, instead functioning as a kind of supplementary parenthetical description of the head NP. Nonetheless, sometimes it is useful to link these phrases to an entity they describe, and so some datasets like ntoNotes mark appositional relationships.

Predicative and Prenominal NPs: Predicative or attributive NPs describe properties of the head noun. In *United is a unit of UAL*, the NP *a unit of UAL* describes a property of *United*, rather than referring to a distinct entity. Thus they are not marked as mentions in coreference tasks; in our example the NPs *\$2.3 million* and *the company's president*, are attributive, describing properties of *her pay* and *the 38-year-old*; Example (22.27) shows a Chinese example in which the predicate NP (中国最大的城市; *China's biggest city*) is not a mention.

- (22.25) her pay jumped to *\$2.3 million*
 (22.26) the 38-year-old became *the company's president*
 (22.27) 上海是[中国最大的城市] [Shanghai is *China's biggest city*]

expletive

clefts

Expletives: Many uses of pronouns like *it* in English and corresponding pronouns in other languages are not referential. Such **expletive** or **pleonastic** cases include *it is raining*, in idioms like *hit it off*, or in particular syntactic situations like **clefts** (22.28a) or **extraposition** (22.28b):

- (22.28) a. *It* was Emma Goldman who founded *Mother Earth*
 b. *It* surprised me that there was a herring hanging on her wall.

Generics: Another kind of expression that does not refer back to an entity explicitly evoked in the text is *generic* reference. Consider (22.29).

- (22.29) I love mangos. *They* are very tasty.

Here, *they* refers, not to a particular mango or set of mangos, but instead to the class of mangos in general. The pronoun *you* can also be used generically:

- (22.30) In July in San Francisco *you* have to wear a jacket.

22.1.4 Linguistic Properties of the Coreference Relation

Now that we have seen the linguistic properties of individual referring expressions we turn to properties of the antecedent/anaphor pair. Understanding these properties is helpful both in designing novel features and performing error analyses.

Number Agreement: Referring expressions and their referents must generally agree in number; English *she/her/he/him/his/it* are singular, *we/us/they/them* are plural, and *you* is unspecified for number. So a plural antecedent like *the chefs* cannot generally corefer with a singular anaphor like *she*. However, algorithms cannot enforce number agreement too strictly. First, semantically plural entities can be referred to by either *it* or *they*:

- (22.31) IBM announced a new machine translation product yesterday. *They* have been working on it for 20 years.

singular they

Second, **singular they** has become much more common, in which *they* is used to describe singular individuals, often useful because *they* is gender neutral. Although recently increasing, singular *they* is old, part of English for many centuries.⁵

Person Agreement: English distinguishes between first, second, and third person, and a pronoun's antecedent must agree with the pronoun in person. Thus a third person pronoun (*he*, *she*, *they*, *him*, *her*, *them*, *his*, *her*, *their*) must have a third person antecedent (one of the above or any other noun phrase). However, phenomena like quotation can cause exceptions; in this example *I*, *my*, and *she* are coreferent:

(22.32) “I voted for Nader because he was most aligned with my values,” she said.

Gender or Noun Class Agreement: In many languages, all nouns have grammatical gender or noun class⁶ and pronouns generally agree with the grammatical gender of their antecedent. In English this occurs only with third-person singular pronouns, which distinguish between *male* (*he*, *him*, *his*), *female* (*she*, *her*), and *nonpersonal* (*it*) grammatical genders. Non-binary pronouns like *ze* or *hir* may also occur in more recent texts. Knowing which gender to associate with a name in text can be complex, and may require world knowledge about the individual. Some examples:

(22.33) Maryam has a theorem. She is exciting. (*she*=Maryam, not the theorem)

(22.34) Maryam has a theorem. It is exciting. (*it*=the theorem, not Maryam)

reflexive

Binding Theory Constraints: The **binding theory** is a name for syntactic constraints on the relations between a mention and an antecedent in the same sentence (Chomsky, 1981). Oversimplifying a bit, **reflexive** pronouns like *himself* and *herself* corefer with the subject of the most immediate clause that contains them (22.35), whereas nonreflexives cannot corefer with this subject (22.36).

(22.35) Janet bought herself a bottle of fish sauce. [*herself*=Janet]

(22.36) Janet bought her a bottle of fish sauce. [*her*≠Janet]

Recency: Entities introduced in recent utterances tend to be more salient than those introduced from utterances further back. Thus, in (22.37), the pronoun *it* is more likely to refer to Jim’s map than the doctor’s map.

(22.37) The doctor found an old map in the captain’s chest. Jim found an even older map hidden on the shelf. It described an island.

Grammatical Role: Entities mentioned in subject position are more salient than those in object position, which are in turn more salient than those mentioned in oblique positions. Thus although the first sentence in (22.38) and (22.39) expresses roughly the same propositional content, the preferred referent for the pronoun *he* varies with the subject—John in (22.38) and Bill in (22.39).

(22.38) Billy Bones went to the bar with Jim Hawkins. He called for a glass of rum. [*he* = Billy]

(22.39) Jim Hawkins went to the bar with Billy Bones. He called for a glass of rum. [*he* = Jim]

⁵ Here’s a bound pronoun example from Shakespeare’s *Comedy of Errors*: *There’s not a man I meet but doth salute me As if I were their well-acquainted friend*

⁶ The word “gender” is generally only used for languages with 2 or 3 noun classes, like most Indo-European languages; many languages, like the Bantu languages or Chinese, have a much larger number of noun classes.

Verb Semantics: Some verbs semantically emphasize one of their arguments, biasing the interpretation of subsequent pronouns. Compare (22.40) and (22.41).

(22.40) John telephoned Bill. He lost the laptop.

(22.41) John criticized Bill. He lost the laptop.

These examples differ only in the verb used in the first sentence, yet “he” in (22.40) is typically resolved to John, whereas “he” in (22.41) is resolved to Bill. This may be due to the link between implicit causality and saliency: the implicit cause of a “criticizing” event is its object, whereas the implicit cause of a “telephoning” event is its subject. In such verbs, the entity which is the implicit cause is more salient.

Selectional Restrictions: Many other kinds of semantic knowledge can play a role in referent preference. For example, the selectional restrictions that a verb places on its arguments (Chapter 20) can help eliminate referents, as in (22.42).

(22.42) I ate the soup in my new bowl after cooking it for hours

There are two possible referents for *it*, the soup and the bowl. The verb *eat*, however, requires that its direct object denote something edible, and this constraint can rule out *bowl* as a possible referent.

22.2 Coreference Tasks and Datasets

We can formulate the task of coreference resolution as follows: Given a text T , find all entities and the coreference links between them. We evaluate our task by comparing the links our system creates with those in human-created gold coreference annotations on T .

Let’s return to our coreference example, now using superscript numbers for each coreference chain (cluster), and subscript letters for individual mentions in the cluster:

(22.43) [Victoria Chen]¹_a, CFO of [Megabucks Banking]²_a, saw [[her]¹_b pay]³_a jump to \$2.3 million, as [the 38-year-old]¹_c also became [[the company]²_b]’s president. It is widely known that [she]¹_d came to [Megabucks]²_c from rival [Lotsabucks]⁴_a.

Assuming example (22.43) was the entirety of the article, the chains for *her pay* and *Lotsabucks* are singleton mentions:

1. {Victoria Chen, her, the 38-year-old, She}
2. {Megabucks Banking, the company, Megabucks}
3. { her pay}
4. { Lotsabucks}

For most coreference evaluation campaigns, the input to the system is the raw text of articles, and systems must detect mentions and then link them into clusters. Solving this task requires dealing with pronominal anaphora (figuring out that *her* refers to *Victoria Chen*), filtering out non-referential pronouns like the pleonastic *It* in *It has been ten years*), dealing with definite noun phrases to figure out that *the 38-year-old* is coreferent with *Victoria Chen*, and that *the company* is the same as *Megabucks*. And we need to deal with names, to realize that *Megabucks* is the same as *Megabucks Banking*.

Exactly what counts as a mention and what links are annotated differs from task to task and dataset to dataset. For example some coreference datasets do not label singletons, making the task much simpler. Resolvers can achieve much higher scores on corpora without singletons, since singletons constitute the majority of mentions in running text, and they are often hard to distinguish from non-referential NPs. Some tasks use gold mention-detection (i.e. the system is given human-labeled mention boundaries and the task is just to cluster these gold mentions), which eliminates the need to detect and segment mentions from running text.

Coreference is usually evaluated by the **CoNLL F1** score, which combines three metrics: MUC, B^3 , and $CEAF_e$; Section 22.7 gives the details.

Let's mention a few characteristics of one popular coreference dataset, OntoNotes (Pradhan et al. 2007, Pradhan et al. 2007a), and the CoNLL 2012 Shared Task based on it (Pradhan et al., 2012a). OntoNotes contains hand-annotated Chinese and English coreference datasets of roughly one million words each, consisting of newswire, magazine articles, broadcast news, broadcast conversations, web data and conversational speech data, as well as about 300,000 words of annotated Arabic newswire. The most important distinguishing characteristic of OntoNotes is that it does not label singletons, simplifying the coreference task, since singletons represent 60%-70% of all entities. In other ways, it is similar to other coreference datasets. Referring expression NPs that are coreferent are marked as mentions, but generics and pleonastic pronouns are not marked. Appositive clauses are not marked as separate mentions, but they are included in the mention. Thus in the NP, “Richard Godown, president of the Industrial Biotechnology Association” the mention is the entire phrase. Prenominal modifiers are annotated as separate entities only if they are proper nouns. Thus *wheat* is not an entity in *wheat fields*, but *UN* is an entity in *UN policy* (but not adjectives like *American* in *American policy*).

A number of corpora mark richer discourse phenomena. The ISNotes corpus annotates a portion of OntoNotes for information status, include bridging examples (Hou et al., 2018). The AnCora-CO coreference corpus (Recasens and Martí, 2010) contains 400,000 words each of Spanish (AnCora-CO-Es) and Catalan (AnCora-CO-Ca) news data, and includes labels for complex phenomena like discourse deixis in both languages. The ARRAU corpus (Uryupina et al., 2019) contains 350,000 words of English marking all NPs, which means singleton clusters are available. ARRAU includes diverse genres like dialog (the TRAINS data) and fiction (the Pear Stories), and has labels for bridging references, discourse deixis, generics, and ambiguous anaphoric relations.

22.3 Mention Detection

mention detection

The first stage of coreference is **mention detection**: finding the spans of text that constitute each mention. Mention detection algorithms are usually very liberal in proposing candidate mentions (i.e., emphasizing recall), and only filtering later. For example many systems run parsers and named entity taggers on the text and extract every span that is either an **NP**, a **possessive pronoun**, or a **named entity**.

Doing so from our sample text repeated in (22.44):

(22.44) Victoria Chen, CFO of Megabucks Banking, saw her pay jump to \$2.3 million, as the 38-year-old also became the company’s president. It is widely known that she came to Megabucks from rival Lotsabucks.

might result in the following list of 13 potential mentions:

Victoria Chen	the company
CFO of Megabucks Banking	the company's president
Megabucks Banking	It
her	she
her pay	Megabucks
\$2.3 million	Lotsabucks
the 38-year-old	

More recent mention detection systems are even more generous; the span-based algorithm we will describe in Section 22.6 first extracts literally all N-gram spans of words up to $N=10$. Of course recall from Section 22.1.3 that many NPs—and the overwhelming majority of random N-gram spans—are not referring expressions. Therefore all such mention detection systems need to eventually filter out pleonastic/expletive pronouns like *It* above, appositives like *CFO of Megabucks Banking Inc*, or predicate nominals like *the company's president* or *\$2.3 million*.

Some of this filtering can be done by rules. Early rule-based systems designed regular expressions to deal with pleonastic *it*, like the following rules from Lappin and Leass (1994) that use dictionaries of cognitive verbs (e.g., *believe*, *know*, *anticipate*) to capture pleonastic *it* in “It is *thought* that ketchup...”, or modal adjectives (e.g., *necessary*, *possible*, *certain*, *important*), for, e.g., “It is *likely* that I...”. Such rules are sometimes used as part of modern systems:

```
It is Modaladjective that S
It is Modaladjective (for NP) to VP
It is Cogv-ed that S
It seems/appears/means/follows (that) S
```

Mention-detection rules are sometimes designed specifically for particular evaluation campaigns. For OntoNotes, for example, mentions are not embedded within larger mentions, and while numeric quantities are annotated, they are rarely coreferential. Thus for OntoNotes tasks like CoNLL 2012 (Pradhan et al., 2012a), a common first pass rule-based mention detection algorithm (Lee et al., 2013) is:

1. Take all NPs, possessive pronouns, and named entities.
2. Remove numeric quantities (100 dollars, 8%), mentions embedded in larger mentions, adjectival forms of nations, and stop words (like *there*).
3. Remove pleonastic *it* based on regular expression patterns.

Rule-based systems, however, are generally insufficient to deal with mention-detection, and so modern systems incorporate some sort of learned mention detection component, such as a **referentiality classifier**, an **anaphoricity classifier**—detecting whether an NP is an anaphor—or a **discourse-new classifier**—detecting whether a mention is discourse-new and a potential antecedent for a future anaphor.

anaphoricity
detector

An **anaphoricity detector**, for example, can draw its positive training examples from any span that is labeled as an anaphoric referring expression in hand-labeled datasets like OntoNotes, ARRAU, or AnCora. Any other NP or named entity can be marked as a negative training example. Anaphoricity classifiers use features of the candidate mention such as its head word, surrounding words, definiteness, animacy, length, position in the sentence/discourse, many of which were first proposed in early work by Ng and Cardie (2002a); see Section 22.5 for more on features.

Referentiality or anaphoricity detectors can be run as filters, in which only mentions that are classified as anaphoric or referential are passed on to the coreference system. The end result of such a filtering mention detection system on our example above might be the following filtered set of 9 potential mentions:

Victoria Chen	her pay	she
Megabucks Bank	the 38-year-old	Megabucks
her	the company	Lotsabucks

It turns out, however, that hard filtering of mentions based on an anaphoricity or referentiality classifier leads to poor performance. If the anaphoricity classifier threshold is set too high, too many mentions are filtered out and recall suffers. If the classifier threshold is set too low, too many pleonastic or non-referential mentions are included and precision suffers.

The modern approach is instead to perform mention detection, anaphoricity, and coreference jointly in a single end-to-end model (Ng 2005b, Denis and Baldridge 2007, Rahman and Ng 2009). For example mention detection in the Lee et al. (2017b), (2018) system is based on a single end-to-end neural network that computes a score for each mention being referential, a score for two mentions being coreference, and combines them to make a decision, training all these scores with a single end-to-end loss. We'll describe this method in detail in Section 22.6.⁷

Despite these advances, correctly detecting referential mentions seems to still be an unsolved problem, since systems incorrectly marking pleonastic pronouns like *it* and other non-referential NPs as coreferent is a large source of errors of modern coreference resolution systems (Kummerfeld and Klein 2013, Martschat and Strube 2014, Martschat and Strube 2015, Wiseman et al. 2015, Lee et al. 2017a).

Mention, referentiality, or anaphoricity detection is thus an important open area of investigation. Other sources of knowledge may turn out to be helpful, especially in combination with unsupervised and semisupervised algorithms, which also mitigate the expense of labeled datasets. In early work, for example Bean and Riloff (1999) learned patterns for characterizing anaphoric or non-anaphoric NPs; (by extracting and generalizing over the first NPs in a text, which are guaranteed to be non-anaphoric). Chang et al. (2012) look for head nouns that appear frequently in the training data but never appear as gold mentions to help find non-referential NPs. Bergsma et al. (2008b) use web counts as a semisupervised way to augment standard features for anaphoricity detection for English *it*, an important task because *it* is both common and ambiguous; between a quarter and half *it* examples are non-anaphoric. Consider the following two examples:

(22.45) You can make [it] in advance. [anaphoric]

(22.46) You can make [it] in Hollywood. [non-anaphoric]

The *it* in *make it* is non-anaphoric, part of the idiom *make it*. Bergsma et al. (2008b) turn the context around each example into patterns, like “make * in advance” from (22.45), and “make * in Hollywood” from (22.46). They then use Google N-grams to enumerate all the words that can replace *it* in the patterns. Non-anaphoric contexts tend to only have *it* in the wildcard positions, while anaphoric contexts occur with many other NPs (for example *make them in advance* is just as frequent in their data

⁷ Some systems try to avoid mention detection or anaphoricity detection altogether. For datasets like OntoNotes which don't label singletons, an alternative to filtering out non-referential mentions is to run coreference resolution, and then simply delete any candidate mentions which were not corefered with another mention. This likely doesn't work as well as explicitly modeling referentiality, and cannot solve the problem of detecting singletons, which is important for tasks like entity linking.

as *make it in advance*, but *make them in Hollywood* did not occur at all). These N-gram contexts can be used as features in a supervised anaphoricity classifier.

22.4 Architectures for Coreference Algorithms

Modern systems for coreference are based on supervised neural machine learning, supervised from hand-labeled datasets like OntoNotes. In this section we overview the various architecture of modern systems, using the categorization of Ng (2010), which distinguishes algorithms based on whether they make each coreference decision in a way that is *entity-based*—representing each entity in the discourse model—or only *mention-based*—considering each mention independently, and whether they use *ranking models* to directly compare potential antecedents. Afterwards, we go into more detail on one state-of-the-art algorithm in Section 22.6.

22.4.1 The Mention-Pair Architecture

mention-pair

mention-pair

We begin with the **mention-pair** architecture, the simplest and most influential coreference architecture, which introduces many of the features of more complex algorithms, even though other architectures perform better. The **mention-pair** architecture is based around a classifier that—as its name suggests—is given a pair of mentions, a candidate anaphor and a candidate antecedent, and makes a binary classification decision: corefering or not.

Let's consider the task of this classifier for the pronoun *she* in our example, and assume the slightly simplified set of potential antecedents in Fig. 22.2.



Figure 22.2 For each pair of a mention (like *she*), and a potential antecedent mention (like *Victoria Chen* or *her*), the mention-pair classifier assigns a probability of a coreference link.

For each prior mention (*Victoria Chen*, *Megabucks Banking*, *her*, etc.), the binary classifier computes a probability: whether or not the mention is the antecedent of *she*. We want this probability to be high for actual antecedents (*Victoria Chen*, *her*, *the 38-year-old*) and low for non-antecedents (*Megabucks Banking*, *her pay*).

Early classifiers used hand-built features (Section 22.5); more recent classifiers use neural representation learning (Section 22.6)

For training, we need a heuristic for selecting training samples; since most pairs of mentions in a document are not coreferent, selecting every pair would lead to a massive overabundance of negative samples. The most common heuristic, from (Soon et al., 2001), is to choose the closest antecedent as a positive example, and all pairs in between as the negative examples. More formally, for each anaphor mention m_i we create

- one positive instance (m_i, m_j) where m_j is the closest antecedent to m_i , and

- a negative instance (m_i, m_k) for each m_k between m_j and m_i

Thus for the anaphor *she*, we would choose *(she, her)* as the positive example and no negative examples. Similarly, for the anaphor *the company* we would choose *(the company, Megabucks)* as the positive example and *(the company, she)* (*the company, the 38-year-old*) (*the company, her pay*) and *(the company, her)* as negative examples.

Once the classifier is trained, it is applied to each test sentence in a clustering step. For each mention i in a document, the classifier considers each of the prior $i - 1$ mentions. In **closest-first** clustering (Soon et al., 2001), the classifier is run right to left (from mention $i - 1$ down to mention 1) and the first antecedent with probability $> .5$ is linked to i . If no antecedent has probably > 0.5 , no antecedent is selected for i . In **best-first** clustering, the classifier is run on all $i - 1$ antecedents and the most probable preceding mention is chosen as the antecedent for i . The transitive closure of the pairwise relation is taken as the cluster.

While the mention-pair model has the advantage of simplicity, it has two main problems. First, the classifier doesn't directly compare candidate antecedents to each other, so it's not trained to decide, between two likely antecedents, which one is in fact better. Second, it ignores the discourse model, looking only at mentions, not entities. Each classifier decision is made completely locally to the pair, without being able to take into account other mentions of the same entity. The next two models each address one of these two flaws.

22.4.2 The Mention-Rank Architecture

The mention ranking model directly compares candidate antecedents to each other, choosing the highest-scoring antecedent for each anaphor.

In early formulations, for mention i , the classifier decide which of the $\{1, \dots, i - 1\}$ prior mentions is the antecedent (Denis and Baldridge, 2008). But suppose i is in fact not anaphoric, and none of the antecedents should be chosen? Such a model would need to run a separate anaphoricity classifier on i . Instead, it turns out to be better to jointly learn anaphoricity detection and coreference together with a single loss (Rahman and Ng, 2009).

So in modern mention-ranking systems, for the i th mention (anaphor), we have an associated random variable y_i ranging over the values $Y(i) = \{1, \dots, i - 1, \epsilon\}$. The value ϵ is a special dummy mention meaning that i does not have an antecedent (i.e., is either discourse-new and starts a new coref chain, or is non-anaphoric).

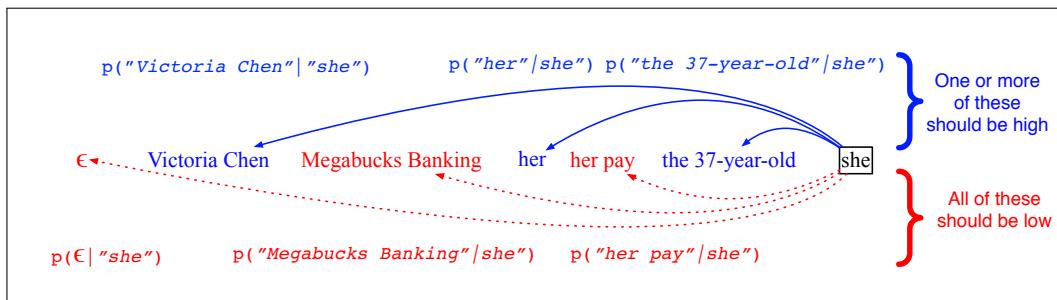


Figure 22.3 For each candidate anaphoric mention (like *she*), the mention-ranking system assigns a probability distribution over all previous mentions plus the special dummy mention ϵ .

At test time, for a given mention i the model computes one softmax over all the antecedents (plus ϵ) giving a probability for each candidate antecedent (or none).

Fig. 22.3 shows an example of the computation for the single candidate anaphor *she*.

Once the antecedent is classified for each anaphor, transitive closure can be run over the pairwise decisions to get a complete clustering.

Training is trickier in the mention-ranking model than the mention-pair model, because for each anaphor we don't know which of all the possible gold antecedents to use for training. Instead, the best antecedent for each mention is *latent*; that is, for each mention we have a whole cluster of legal gold antecedents to choose from. Early work used heuristics to choose an antecedent, for example choosing the closest antecedent as the gold antecedent and all non-antecedents in a window of two sentences as the negative examples (Denis and Baldridge, 2008). Various kinds of ways to model latent antecedents exist (Fernandes et al. 2012, Chang et al. 2013, Durrett and Klein 2013). The simplest way is to give credit to any legal antecedent by summing over all of them, with a loss function that optimizes the likelihood of all correct antecedents from the gold clustering (Lee et al., 2017b). We'll see the details in Section 22.6.

Mention-ranking models can be implemented with hand-build features or with neural representation learning (which might also incorporate some hand-built features). we'll explore both directions in Section 22.5 and Section 22.6.

22.4.3 Entity-based Models

Both the mention-pair and mention-ranking models make their decisions about *mentions*. By contrast, entity-based models link each mention not to a previous mention but to a previous discourse *entity* (cluster of mentions).

A mention-ranking model can be turned into an entity-ranking model simply by having the classifier make its decisions over clusters of mentions rather than individual mentions (Rahman and Ng, 2009).

For traditional feature-based models, this can be done by extracting features over clusters. The size of a cluster is a useful features, as is its 'shape', which is the list of types of the mentions in the cluster i.e., sequences of the tokens (P)roper, (D)efinite, (I)ndefinite, (Pr)onoun, so that a cluster composed of *{Victoria, her, the 38-year-old}* would have the shape *P-Pr-D* (Björkelund and Kuhn, 2014). An entity-based model that includes a mention-pair classifier can use as features aggregates of mention-pair probabilities, for example computing the average probability of coreference over all mention-pairs in the two clusters (Clark and Manning 2015).

Neural models can learn representations of clusters automatically, for example by using an RNN over the sequence of cluster mentions to encode a state corresponding to a cluster representation (Wiseman et al., 2016), or by learning distributed representations for pairs of clusters by pooling over learned representations of mention pairs (Clark and Manning, 2016b).

However, although entity-based models are more expressive, the use of cluster-level information in practice has not led to large gains in performance, so mention-ranking models are still more commonly used.

22.5 Classifiers using hand-built features

Hand-designed features play an important role in coreference, whether as the sole input to classification in pre-neural classifiers, or as augmentations to the automatic

representation learning used in state-of-the-art neural systems like the one we'll describe in Section 22.6.

In this section we describe features commonly used in logistic regression, SVM, or random forest classifiers for coreference resolution.

Given an anaphor mention and a potential antecedent mention, most feature based classifiers make use of three types of features: (i) features of the anaphor, (ii) features of the candidate antecedent, and (iii) features of the relationship between the pair. Entity-based models can make additional use of two additional classes: (iv) feature of all mentions from the antecedent's entity cluster, and (v) features of the relation between the anaphor and the mentions in the antecedent entity cluster.

Figure 22.4 shows a selection of commonly used features, and shows the value that would be computed for the potential anaphor "she" and potential antecedent "Victoria Chen" in our example sentence, repeated below:

(22.47) **Victoria Chen**, CFO of Megabucks Banking, saw her pay jump to \$2.3 million, as the 38-year-old also became the company's president. It is widely known that **she** came to Megabucks from rival Lotsabucks.

Features that prior work has found to be particularly useful are exact string match, entity headword agreement, mention distance, as well as (for pronouns) exact attribute match and i-within-i, and (for nominals and proper names) word inclusion and cosine. For lexical features (like head words) it is common to only use words that appear enough times (perhaps more than 20 times), backing off to parts of speech for rare words.

It is crucial in feature-based systems to use conjunctions of features; one experiment suggested that moving from individual features in a classifier to conjunctions of multiple features increased F1 by 4 points (Lee et al., 2017a). Specific conjunctions can be designed by hand (Durrett and Klein, 2013), all pairs of features can be conjoined (Bengtson and Roth, 2008), or feature conjunctions can be learned automatically, either by using classifiers like decision trees or random forests (Ng and Cardie, 2002a), Lee et al. 2017a) or by using neural models to take raw, unconjoined features as input, and automatically learn intermediate representations (Wiseman et al., 2015).

Finally, some of these features can also be used in neural models as well. Neural systems of the kind we describe in the next section make use of contextual word embeddings, so they don't benefit from adding shallow features like string or head match, grammatical role, or mention types. However features like mention length, distance between mentions, or genre can complement contextual word embedding models nicely.

22.6 A neural mention-ranking algorithm

In this section we describe the neural mention-ranking system of Lee et al. (2017b). This end-to-end system doesn't exactly have a separate mention-detection step. Instead, it considers every possible span of text up to a set length (i.e. all n-grams of length 1,2,3...N) as a possible mention.⁸

⁸ But because this number of potential mentions makes the algorithm very slow and unwieldy (the model's size is $O(t^4)$ in document length) in practice various versions of the algorithm find ways to prune the possible mentions, essentially using a mention score as something of a mention-detector.

Features of the Anaphor or Antecedent Mention		
First (last) word	Victoria/she	First or last word (or embedding) of antecedent/anaphor
Head word	Victoria/she	Head word (or head embedding) of antecedent/anaphor
Attributes	Sg-F-A-3- PER/Sg-F-A- 3-PER	The number, gender, animacy, person, named entity type attributes of (antecedent/anaphor)
Length	2/1	length in words of (antecedent/anaphor)
Grammatical role	Sub/Sub	The grammatical role—subject, direct object, indirect object/PP—of (antecedent/anaphor)
Mention type	P/Pr	Type: (P)roper, (D)efinite, (I)ndefinite, (Pr)onoun) of an- tecedent/anaphor
Features of the Antecedent Entity		
Entity shape	P-Pr-D	The ‘shape’ or list of types of the mentions in the antecedent entity (cluster), i.e., sequences of (P)roper, (D)efinite, (I)ndefinite, (Pr)onoun.
Entity attributes	Sg-F-A-3- PER	The number, gender, animacy, person, named entity type attributes of the antecedent entity
Antecedent cluster size	3	Number of mentions in the antecedent cluster
Features of the Pair of Mentions		
Longer anaphor	F	True if anaphor is longer than antecedent
Pairs of any features	Victoria/she, 2/1, Sub/Sub, P/Pr, etc .	For each individual feature, pair of type of antecedent+ type of anaphor
Sentence distance	1	The number of sentences between antecedent and anaphor
Mention distance	4	The number of mentions between antecedent and anaphor
i-within-i	F	Anaphor has i-within-i relation with antecedent
Cosine		Cosine between antecedent and anaphor embeddings
Appositive	F	True if the anaphor is in the syntactic apposition relation to the antecedent. This can be useful even if appositives are not mentions (to know to attach the appositive to a preceding head)
Features of the Pair of Entities		
Exact String Match	F	True if the strings of any two mentions from the antecedent and anaphor clusters are identical.
Head Word Match	F	True if any mentions from antecedent cluster has same headword as any mention in anaphor cluster
Word Inclusion	F	Words in antecedent cluster includes all words in anaphor cluster
Features of the Document		
Genre/source	N	The document genre—(D)ialog, (N)ews, etc,

Figure 22.4 Some common features for feature-based coreference algorithms, with values for the anaphor “she” and potential antecedent “Victoria Chen”.

Given a document D with T words, the model considers all of the $N = \frac{T(T-1)}{2}$ text spans up to some length (in the version of Lee et al. (2018), that length is 10). Each span i starts at word $\text{START}(i)$ and ends at word $\text{END}(i)$.

The task is to assign to each span i an antecedent y_i , a random variable ranging over the values $Y(i) = \{1, \dots, i-1, \epsilon\}$; each previous span and a special dummy token ϵ . Choosing the dummy token means that i does not have an antecedent, either

because i is discourse-new and starts a new coreference chain, or because i is non-anaphoric.

For each pair of spans i and j , the system assigns a score $s(i, j)$ for the coreference link between span i and span j . The system then learns a distribution $P(y_i)$ over the antecedents for span i :

$$P(y_i) = \frac{\exp(s(i, y_i))}{\sum_{y' \in Y(i)} \exp(s(i, y_i))} \quad (22.48)$$

This score $s(i, j)$ includes three factors: $m(i)$; whether span i is a mention; $m(j)$; whether span j is a mention; and $c(j)$; whether j is the antecedent of i :

$$s(i, j) = m(i) + m(j) + c(i, j) \quad (22.49)$$

For the dummy antecedent ϵ , the score $s(i, \epsilon)$ is fixed to 0. This way if any non-dummy scores are positive, the model predicts the highest-scoring antecedent, but if all the scores are negative it abstains.

The scoring functions $m(i)$ and $c(i, j)$ are based on a vector \mathbf{g}_i that represents span i :

$$m(i) = w_m \cdot \text{FFNN}_m(\mathbf{g}_i) \quad (22.50)$$

$$c(i, j) = w_c \cdot \text{FFNN}_c([\mathbf{g}_i, \mathbf{g}_j, \mathbf{g}_i \circ \mathbf{g}_j, \phi(i, j)]) \quad (22.51)$$

The antecedent score $c(i, j)$ takes as input a representation of the spans i and j , but also the element-wise similarity of the two spans to each other $\mathbf{g}_i \circ \mathbf{g}_j$ (here \circ is element-wise multiplication). The antecedent score c also considers a feature vector $\phi(i, j)$ that encodes useful features like mention distances, and also information about the speaker and genre.

The span representations \mathbf{g}_i themselves consist of two parts: a contextual representation of the first and last word in the span, and a representation of the headword of the span. The contextual representations of the first and last words of each span are computed by a standard biLSTM. The biLSTM takes as input a representation w_t for each word, based on contextual word embeddings like ELMo. (Using BERT instead of ELMo results in even higher performance (Joshi et al., 2019)). The output of the biLSTM for each word w_t of the input is \mathbf{h}_t :

$$\begin{aligned} \overrightarrow{\mathbf{h}}_t &= \text{LSTM}^{\text{forward}}(\overleftarrow{\mathbf{h}}_{t-1}, \mathbf{w}_t) \\ \overleftarrow{\mathbf{h}}_t &= \text{LSTM}^{\text{backward}}(\overrightarrow{\mathbf{h}}_{t+1}, \mathbf{w}_t) \\ \mathbf{h}_t &= [\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t] \end{aligned} \quad (22.52)$$

The system uses independent LSTMs for each sentence.

The system uses attention (Chapter 10) over the words in the span to represent the span's head. As is usual with attention, the system learns a weight vector \mathbf{w}_α , and computes its dot product with the hidden state \mathbf{h}_t transformed by a FFNN:

$$\alpha_t = \mathbf{w}_\alpha \cdot \text{FFNN}_\alpha(\mathbf{h}_t) \quad (22.53)$$

The attention score is normalized into a distribution via a softmax:

$$a_{i,t} = \frac{\exp(\alpha_t)}{\sum_{k=\text{START}(i)}^{\text{END}(i)} \exp(\alpha_k)} \quad (22.54)$$

And then the attention distribution is used to create a vector $\mathbf{h}_{\text{ATT}(i)}$ which is an attention-weighted sum of words in span i :

$$\mathbf{h}_{\text{ATT}(i)} = \sum_{t=\text{START}(i)}^{\text{END}(i)} a_{i,t} \cdot \mathbf{w}_t \quad (22.55)$$

Each span i is then represented by a vector \mathbf{g}_i , a concatenation of the hidden representations of the start and end tokens of the span, the head, and a feature vector containing only one feature: the length of span i .

$$\mathbf{g}_i = [\mathbf{h}_{\text{START}(i)}, \mathbf{h}_{\text{END}(i)}, \mathbf{h}_{\text{ATT}(i)}, \phi(i)] \quad (22.56)$$

Fig. 22.5 from Lee et al. (2017b) shows the computation of the span representation and the mention score.

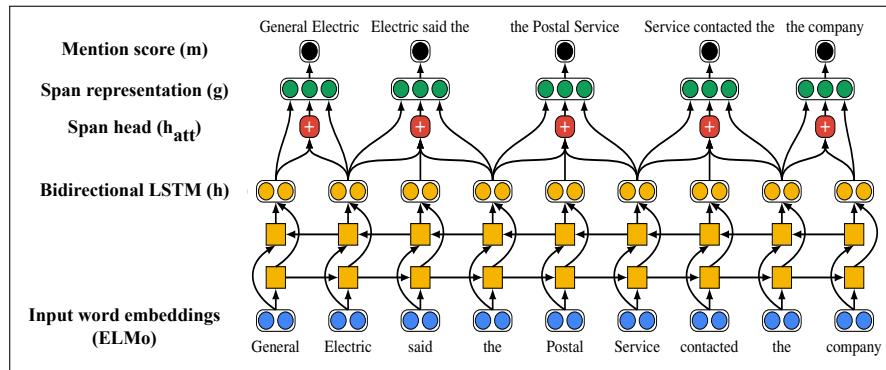


Figure 22.5 Computation of the span representation and the mention score in the end-to-end coreference model of Lee et al. (2017b). The model considers all spans up to a maximum width; the figure shows a small subset of these. Figure after Lee et al. (2017b).

Fig. 22.6 shows the computation of the score s for the three possible antecedents of *the company* in the example sentence from Fig. 22.5.

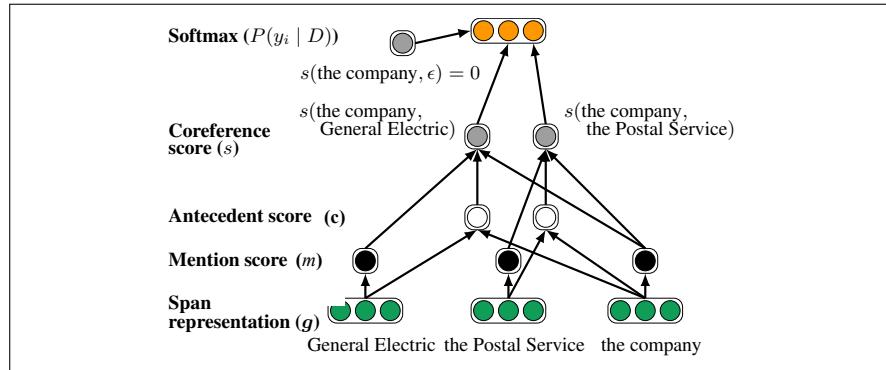


Figure 22.6 The computation of the score s for the three possible antecedents of *the company* in the example sentence from Fig. 22.5. Figure after Lee et al. (2017b).

At inference time, some method is generally used to prune the mentions (for example using the mention score m as a filter to keep only the best few mentions as a function like $0.4T$ of the sentence length T). Then the joint distribution of

antecedents for each document is computed in a forward pass. Finally, we can then do transitive closure on the antecedents to create a final clustering for the document.

For training, we don't have a single gold antecedent for each mention; instead the coreference labeling only gives us each entire cluster of coreferent mentions, and a mention has a latent antecedent. We therefore use a loss function that maximizes the sum of the coreference probability of any of the legal antecedents. For a given mention i with possible antecedents $Y(i)$, let $\text{GOLD}(i)$ be the set of mentions in the gold cluster containing i . Since the set of mentions occurring before i is $Y(i)$, the set of mentions in that gold cluster that also occur before i is $Y(i) \cap \text{GOLD}(i)$. We therefore want to maximize:

$$\sum_{\hat{y} \in Y(i) \cap \text{GOLD}(i)} P(\hat{y}) \quad (22.57)$$

If a mention i is not in a gold cluster $\text{GOLD}(i) = \emptyset$.

To turn this probability into a loss function, we'll use the cross-entropy loss function we defined in Eq. 5.10 in Chapter 5, by taking the $-\log$ of the probability. If we then sum over all mentions, we get the final loss function for training:

$$L = \sum_{i=2}^N -\log \sum_{\hat{y} \in Y(i) \cap \text{GOLD}(i)} P(\hat{y}) \quad (22.58)$$

Fig. 22.7 shows example predictions from the model, showing the attention weights, which Lee et al. (2017b) find correlate with traditional semantic heads. Note that the model gets the second example wrong, presumably because *attendants* and *pilot* likely have nearby word embeddings.

We are looking for (a **region** of central Italy bordering the Adriatic Sea). (**The area**) is mostly mountainous and includes Mt. Corno, the highest peak of the Apennines. (**It**) also includes a lot of sheep, good clean-living, healthy sheep, and an Italian entrepreneur has an idea about how to make a little money of them.

(**The flight attendants**) have until 6:00 today to ratify labor concessions. (**The pilots'**) union and ground crew did so yesterday.

Figure 22.7 Sample predictions from the Lee et al. (2017b) model, with one cluster per example, showing one correct example and one mistake. Bold, parenthesized spans are mentions in the predicted cluster. The amount of red color on a word indicates the head-finding attention weight $a_{i,t}$ in (22.54). Figure adapted from Lee et al. (2017b).

22.7 Evaluation of Coreference Resolution

We evaluate coreference algorithms model-theoretically, comparing a set of **hypothesis** chains or clusters H produced by the system against a set of gold or **reference** chains or clusters R from a human labeling, and reporting precision and recall.

However, there are a wide variety of methods for doing this comparison. In fact, there are 5 common metrics used to evaluate coreference algorithms: the **link** based MUC (Vilain et al., 1995) and BLANC (Recasens and Hovy 2011, Luo et al. 2014) metrics, the **mention** based B^3 metric (Bagga and Baldwin, 1998), the **entity** based CEAF metric (Luo, 2005), and the **link** based **entity** aware LEA metric (Moosavi and Strube, 2016).

MUC F-measure

Let's just explore two of the metrics. The **MUC F-measure** (Vilain et al., 1995) is based on the number of coreference *links* (pairs of mentions) common to H and R . Precision is the number of common links divided by the number of links in H . Recall is the number of common links divided by the number of links in R ; This makes MUC biased toward systems that produce large chains (and fewer entities), and it ignores singletons, since they don't involve links.

B³

B^3 is mention-based rather than link-based. For each mention in the reference chain, we compute a precision and recall, and then we take a weighted sum over all N mentions in the document to compute a precision and recall for the entire task. For a given mention i , let R be the reference chain that includes i , and H the hypothesis chain that has i . The set of correct mentions in H is $H \cap R$. Precision for mention i is thus $\frac{|H \cap R|}{|H|}$, and recall for mention i thus $\frac{|H \cap R|}{|R|}$. The total precision is the weighted sum of the precision for mention i , weighted by a weight w_i . The total recall is the weighted sum of the recall for mention i , weighted by a weight w_i . Equivalently:

$$\text{Precision} = \sum_{i=1}^N w_i \frac{\# \text{ of correct mentions in hypothesis chain containing entity}_i}{\# \text{ of mentions in hypothesis chain containing entity}_i}$$

$$\text{Recall} = \sum_{i=1}^N w_i \frac{\# \text{ of correct mentions in hypothesis chain containing entity}_i}{\# \text{ of mentions in reference chain containing entity}_i}$$

The weight w_i for each entity can be set to different values to produce different versions of the algorithm.

Following a proposal from Denis and Baldridge (2009), the CoNLL coreference competitions were scored based on the average of MUC, CEAF-e, and B^3 (Pradhan et al. 2011, Pradhan et al. 2012b), and so it is common in many evaluation campaigns to report an average of these 3 metrics. See Luo and Pradhan (2016) for a detailed description of the entire set of metrics; reference implementations of these should be used rather than attempting to reimplement from scratch (Pradhan et al., 2014).

Alternative metrics have been proposed that deal with particular coreference domains or tasks. For example, consider the task of resolving mentions to named entities (persons, organizations, geopolitical entities), which might be useful for information extraction or knowledge base completion. A hypothesis chain that correctly contains all the pronouns referring to an entity, but has no version of the name itself, or is linked with a wrong name, is not useful for this task. We might instead want a metric that weights each mention by how informative it is (with names being most informative) (Chen and Ng, 2013) or a metric that considers a hypothesis to match a gold chain only if it contains at least one variant of a name (the NEC F1 metric of Agarwal et al. (2019)).

22.8 Entity Linking

entity linking

The task of **entity linking** (Ji and Grishman, 2011), closely related to coreference, is to associate a mention in text with the representation of some real-world entity in an ontology, a list of entities in the world, like a gazeteer (Chapter 16). Perhaps the most common ontology used for this task is Wikipedia, in which each Wikipedia page acts as the unique id for a particular entity. Thus the entity linking task of **wikification** (Mihalcea and Csomai, 2007) is the task of deciding which Wikipedia page corresponding to an individual is being referred to by a mention. We'll consider

wikification

that task for the rest of this section, but see Ling et al. (2015b) on different linking tasks and datasets.

anchor texts

Since the earliest systems (Mihalcea and Csomai 2007, Cucerzan 2007, Milne and Witten 2008), entity linking is done in two stages: **mention detection** and **mention disambiguation**. A very useful feature for mention detection is what Mihalcea and Csomai (2007) called a **key phrase**: the mapping between Wikipedia **anchor texts** (the hyperlinked span of text associated with a URL, like *Stanford University*, *Stanford*, or *Governor Stanford*) and the Wikipedia page title it links to (*Stanford University*, or *Leland Stanford*). Prebuilt dictionaries of these anchor text/title page links are available (Spitkovsky and Chang, 2012). Mention detection steps also often include various kinds of query expansion, for example by doing coreference resolution on the current document. Mention disambiguation is often done by supervised learning

Coreference can help entity linking, by giving more possible surface forms to help link to the right Wikipedia page. But entity linking can also be used in the other direction, to improve coreference resolution. Consider this example from Hajishirzi et al. (2013):

- (22.59) [Michael Eisner]₁ and [Donald Tsang]₂ announced the grand opening of [[Hong Kong]₃ Disneyland]₄ yesterday. [Eisner]₁ thanked [the President]₂ and welcomed [fans]₅ to [the park]₄.

Integrating entity linking into coreference can help draw encyclopedic knowledge (like the fact that *Donald Tsang* is a president) to help disambiguate the mention *the President*. Ponzetto and Strube (2006) (2007) and Ratinov and Roth (2012) showed that such attributes extracted from Wikipedia pages could be used to build richer models of entity mentions in coreference. More recent research shows how to do linking and coreference jointly (Hajishirzi et al. 2013, Zheng et al. 2013) or even jointly with named entity tagging as well (Durrett and Klein 2014).

22.9 Winograd Schema problems

From early on in the field, researchers have noted that some cases of coreference are quite difficult, seeming to require world knowledge or sophisticated reasoning to solve. The problem was most famously pointed out by Winograd (1972) with the following example:

- (22.60) The city council denied the demonstrators a permit because
- they feared violence.
 - they advocated violence.

Winograd noticed that the antecedent that most readers preferred for the pronoun *they* in continuation (a) was *the city council*, but in (b) was *the demonstrators*. He suggested that this requires understanding that the second clause is intended as an explanation of the first clause, and also that our cultural frames suggest that city councils are perhaps more likely than demonstrators to fear violence and that demonstrators might be more likely to advocate violence.

Winograd schema

In an attempt to get the field of NLP to focus more on methods involving world knowledge and common sense reasoning, Levesque (2011) proposed a challenge task called the **Winograd Schema Challenge**.⁹ The problems in the challenge task

are coreference problems designed to be easily disambiguated by the human reader, but hopefully not solvable by simple techniques such as selectional restrictions, or other basic word association methods.

The problems are framed as a pair of statements that differ in a single word or phrase, and a coreference question:

- (22.61) The trophy didn't fit into the suitcase because it was too **large**.

Question: What was too **large**? Answer: The trophy

- (22.62) The trophy didn't fit into the suitcase because it was too **small**.

Question: What was too **small**? Answer: The suitcase

The problems have the following characteristics:

1. The problems each have two parties
2. A pronoun preferentially refers to one of the parties, but could grammatically also refer to the other
3. A question asks which party the pronoun refers to
4. If one word in the question is changed, the human-preferred answer changes to the other party

The kind of world knowledge that might be needed to solve the problems can vary. In the trophy/suitcase example, it is knowledge about the physical world; that a bigger object cannot fit into a smaller object. In the original Winograd sentence, it is stereotypes about social actors like politicians and protesters. In examples like the following, it is knowledge about human actions like turn-taking or thanking.

- (22.63) Bill passed the gameboy to John because his turn was [over/next]. Whose turn was [over/next]? Answers: Bill/John

- (22.64) Joan made sure to thank Susan for all the help she had [given/received]. Who had [given/received] help? Answers: Susan/Joan.

Although the Winograd Schema was designed to require common-sense reasoning, a large percentage of the original set of problem can be solved by pre-trained language models, fine-tuned on Winograd Schema sentences (Kocijan et al., 2019). Large pre-trained language models encode an enormous amount of world or common-sense knowledge! The current trend is therefore to propose new datasets with increasingly difficult Winograd-like coreference resolution problems like KNOWREF (Emami et al., 2019), with examples like:

- (22.65) Marcus is undoubtedly faster than Jarrett right now but in [his] prime the gap wasn't all that big.

In the end, it seems likely that some combination of language modeling and knowledge will prove fruitful; indeed, it seems that knowledge-based models overfit less to lexical idiosyncrasies in Winograd Schema training sets (Trichelair et al., 2018),

22.10 Gender Bias in Coreference

As with other aspects of language processing, coreference models exhibit gender and other biases (Zhao et al. 2018a, Rudinger et al. 2018, Webster et al. 2018).

⁹ Levesque's call was quickly followed up by Levesque et al. (2012) and Rahman and Ng (2012), a competition at the IJCAI conference (Davis et al., 2017), and a natural language inference version of the problem called WNLI (Wang et al., 2018).

For example the WinoBias dataset (Zhao et al., 2018a) uses a variant of the Winograd Schema paradigm to test the extent to which coreference algorithms are biased toward linking gendered pronouns with antecedents consistent with cultural stereotypes. As we summarized in Chapter 6, embeddings replicate societal biases in their training test, such as associating men with historically stereotypical male occupations like doctors, and women with stereotypical female occupations like secretaries (Caliskan et al. 2017, Garg et al. 2018).

A WinoBias sentence contain two mentions corresponding to stereotypically-male and stereotypically-female occupations and a gendered pronoun that must be linked to one of them. The sentence cannot be disambiguated by the gender of the pronoun, but a biased model might be distracted by this cue. Here is an example sentence:

- (22.66) The secretary called the physician_i and told him_i about a new patient [pro-stereotypical]
- (22.67) The secretary called the physician_i and told her_i about a new patient [anti-stereotypical]

Zhao et al. (2018a) consider a coreference system to be biased if it is more accurate at linking pronouns consistent with gender stereotypical occupations (e.g., *him* with *physician* in (22.66)) than linking pronouns inconsistent with gender-stereotypical occupations (e.g., *her* with *physician* in (22.67)). They show that coreference systems of all architectures (rule-based, feature-based machine learned, and end-to-end-neural) all show significant bias, performing on average 21 F₁ points worse in the anti-stereotypical cases.

One possible source of this bias is that female entities are significantly underrepresented in the OntoNotes dataset, used to train most coreference systems. Zhao et al. (2018a) propose a way to overcome this bias: they generate a second gender-swapped dataset in which all male entities in OntoNotes are replaced with female ones and vice versa, and retrain coreference systems on the combined original and swapped OntoNotes data, also using debiased GloVe embeddings (Bolukbasi et al., 2016). The resulting coreference systems no longer exhibit bias on the WinoBias dataset, without significantly impacting OntoNotes coreference accuracy. In a follow-up paper, Zhao et al. (2019) show that the same biases exist in ELMo contextualized word vector representations and coref systems that use them. They showed that retraining ELMo with data augmentation again reduces or removes bias in coreference systems on WinoBias.

Webster et al. (2018) introduces another dataset, GAP, and the task of Gendered Pronoun Resolution as a tool for developing improved coreference algorithms for gendered pronouns. GAP is a gender-balanced labeled corpus of 4,454 sentences with gendered ambiguous pronouns (by contrast, only 20% of the gendered pronouns in the English OntoNotes training data are feminine). The examples were created by drawing on naturally occurring sentences from Wikipedia pages to create hard to resolve cases with two named entities of the same gender and an ambiguous pronoun that may refer to either person (or neither), like the following:

- (22.68) In May, Fujisawa joined Mari Motohashi’s rink as the team’s skip, moving back from Karuizawa to Kitami where she had spent her junior days.

Webster et al. (2018) show that modern coreference algorithms perform significantly worse on resolving feminine pronouns than masculine pronouns in GAP. Kurita et al. (2019) shows that a system based on BERT contextualized word representations shows similar bias.

22.11 Summary

This chapter introduced the task of **coreference resolution**.

- This is the task of linking together **mentions** in text which **corefer**, i.e. refer to the same **discourse entity** in the **discourse model**, resulting in a set of coreference **chains** (also called **clusters** or **entities**).
- Mentions can be **definite NPs** or **indefinite NPs**, **pronouns** (including **zero pronouns**) or **names**.
- The surface form of an entity mention is linked to its **information status** (**new**, **old**, or **inferrable**), and how **accessible** or **salient** the entity is.
- Some NPs are not referring expressions, such as pleonastic *it* in *It is raining*.
- Many corpora have human-labeled coreference annotations that can be used for supervised learning, including **OntoNotes** for English, Chinese, and Arabic, **ARRAU** for English, and **AnCora** for Spanish and Catalan.
- Mention detection can start with all nouns and named entities and then use **anaphoricity classifiers** or **referentiality classifiers** to filter out non-mentions.
- Three common architectures for coreference are **mention-pair**, **mention-rank**, and **entity-based**, each of which can make use of feature-based or neural classifiers.
- Modern coreference systems tend to be end-to-end, performing mention detection and coreference in a single end-to-end architecture.
- Algorithms learn representations for text spans and heads, and learn to compare anaphor spans with candidate antecedent spans.
- Coreference systems are evaluated by comparing with gold entity labels using precision/recall metrics like **MUC**, **B³**, **CEAF**, **BLANC**, or **LEA**.
- The **Winograd Schema Challenge** problems are difficult coreference problems that seem to require world knowledge or sophisticated reasoning to solve.
- Coreference systems exhibit **gender bias** which can be evaluated using datasets like Winobias and GAP.

Bibliographical and Historical Notes

Coreference has been part of natural language understanding since the 1970s (Woods et al. 1972, Winograd 1972). The discourse model and the entity-centric foundation of coreference was formulated by Karttunen (1969) (at the 3rd COLING conference), playing a role also in linguistic semantics (Heim 1982, Kamp 1981). But it was Bonnie Webber’s (1978) dissertation and following work (Webber 1983) that explored the model’s computational aspects, providing fundamental insights into how entities are represented in the discourse model and the ways in which they can license subsequent reference. Many of the examples she provided continue to challenge theories of reference to this day.

Hobbs algorithm

The **Hobbs algorithm**¹⁰ is a tree-search algorithm that was the first in a long series of syntax-based methods for identifying reference robustly in naturally occurring text. The input to the Hobbs algorithm is a pronoun to be resolved, together

¹⁰ The simpler of two algorithms presented originally in Hobbs (1978).

with a syntactic (constituency) parse of the sentences up to and including the current sentence. The details of the algorithm depend on the grammar used, but can be understood from a simplified version due to [Kehler et al. \(2004\)](#) that just searches through the list of NPs in the current and prior sentences. This simplified Hobbs algorithm searches NPs in the following order: “(i) in the current sentence from right-to-left, starting with the first NP to the left of the pronoun, (ii) in the previous sentence from left-to-right, (iii) in two sentences prior from left-to-right, and (iv) in the current sentence from left-to-right, starting with the first noun group to the right of the pronoun (for cataphora). The first noun group that agrees with the pronoun with respect to number, gender, and person is chosen as the antecedent” ([Kehler et al., 2004](#)).

[Lappin and Leass \(1994\)](#) was an influential entity-based system that used weights to combine syntactic and other features, extended soon after by [Kennedy and Boguraev \(1996\)](#) whose system avoids the need for full syntactic parses.

Approximately contemporaneously centering ([Grosz et al., 1995](#)) was applied to pronominal anaphora resolution by [Brennan et al. \(1987\)](#), and a wide variety of work followed focused on centering’s use in coreference ([Kameyama 1986, Di Eugenio 1990, Walker et al. 1994, Di Eugenio 1996, Strube and Hahn 1996, Kehler 1997a, Tetreault 2001, Iida et al. 2003](#)). [Kehler and Rohde \(2013\)](#) show how centering can be integrated with coherence-driven theories of pronoun interpretation. See Chapter 23 for the use of centering in measuring discourse coherence.

Coreference competitions as part of the US DARPA-sponsored MUC conferences provided early labeled coreference datasets (the 1995 MUC-6 and 1998 MUC-7 corpora), and set the tone for much later work, choosing to focus exclusively on the simplest cases of *identity coreference* (ignoring difficult cases like bridging, metonymy, and part-whole) and drawing the community toward supervised machine learning and metrics like the MUC metric ([Vilain et al., 1995](#)). The later ACE evaluations produced labeled coreference corpora in English, Chinese, and Arabic that were widely used for model training and evaluation.

This DARPA work influenced the community toward supervised learning beginning in the mid-90s ([Connolly et al. 1994, Aone and Bennett 1995, McCarthy and Lehnert 1995](#)). [Soon et al. \(2001\)](#) laid out a set of basic features, extended by [Ng and Cardie \(2002b\)](#), and a series of machine learning models followed over the next 15 years. These often focused separately on pronominal anaphora resolution ([Kehler et al. 2004, Bergsma and Lin 2006](#)), full NP coreference ([Cardie and Wagstaff 1999, Ng and Cardie 2002b, Ng 2005a](#)) and definite NP reference ([Poesio and Vieira 1998, Vieira and Poesio 2000](#)), as well as separate anaphoricity detection ([Bean and Riloff 1999, Bean and Riloff 2004, Ng and Cardie 2002a, Ng 2004](#)), or singleton detection ([de Marneffe et al., 2015](#)).

The move from mention-pair to mention-ranking approaches was pioneered by [Yang et al. \(2003\)](#) and [Iida et al. \(2003\)](#) who proposed pairwise ranking methods, then extended by [Denis and Baldridge \(2008\)](#) who proposed to do ranking via a softmax over all prior mentions. The idea of doing mention detection, anaphoricity, and coreference jointly in a single end-to-end model grew out of the early proposal of [Ng \(2005b\)](#) to use a dummy antecedent for mention-ranking, allowing ‘non-referential’ to be a choice for coreference classifiers, Denis and Baldridge’s (2007) joint system combining anaphoricity classifier probabilities with coreference probabilities, the [Denis and Baldridge \(2008\)](#) ranking model, and the [Rahman and Ng \(2009\)](#) proposal to train the two models jointly with a single objective.

Simple rule-based systems for coreference returned to prominence in the 2010s,

partly because of their ability to encode entity-based features in a high-precision way (Zhou et al. 2004, Haghghi and Klein 2009, Raghunathan et al. 2010, Lee et al. 2011, Lee et al. 2013, Hajishirzi et al. 2013) but in the end they suffered from an inability to deal with the semantics necessary to correctly handle cases of common noun coreference.

A return to supervised learning led to a number of advances in mention-ranking models which were also extended into neural architectures, for example using reinforcement learning to directly optimize coreference evaluation models Clark and Manning (2016a), doing end-to-end coreference all the way from span extraction (Lee et al. 2017b, Zhang et al. 2018). Neural models also were designed to take advantage of global entity-level information (Clark and Manning 2016b, Wiseman et al. 2016, Lee et al. 2018).

metonymy

The coreference task as we introduced it involves a simplifying assumption that the relationship between an anaphor and its antecedent is one of *identity*: the two corefering mentions refer to the identical discourse referent. In real texts, the relationship can be more complex, where different aspects of a discourse referent can be neutralized or refocused. For example (22.69) (Recasens et al., 2011) shows an example of **metonymy**, in which the capital city *Washington* is used metonymically to refer to the US. (22.70-22.71) show other examples (Recasens et al., 2011):

- (22.69) a strict interpretation of a policy requires **The U.S.** to notify foreign dictators of certain coup plots ... **Washington** rejected the bid ...
- (22.70) I once crossed that border into Ashgh-Abad on Nowruz, the Persian New Year. In the South, everyone was celebrating **New Year**; to the North, **it** was a regular day.
- (22.71) In France, **the president** is elected for a term of seven years, while in the United States **he** is elected for a term of four years.

For further linguistic discussions of these complications of coreference see Pustejovsky (1991), van Deemter and Kibble (2000), Poesio et al. (2006), Fauconnier and Turner (2008), Versley (2008), and Barker (2010).

Ng (2017) offers a useful compact history of machine learning models in coreference resolution. There are three excellent book-length surveys of anaphora/coreference resolution, covering different time periods: Hirst (1981) (early work until about 1981), Mitkov (2002) (1986-2001), and Poesio et al. (2016) (2001-2015).

Exercises

Discourse Coherence

And even in our wildest and most wandering reveries, nay in our very dreams, we shall find, if we reflect, that the imagination ran not altogether at adventures, but that there was still a connection upheld among the different ideas, which succeeded each other. Were the loosest and freest conversation to be transcribed, there would immediately be transcribed, there would immediately be observed something which connected it in all its transitions.

David Hume, *An enquiry concerning human understanding*, 1748

Orson Welles' movie *Citizen Kane* was groundbreaking in many ways, perhaps most notably in its structure. The story of the life of fictional media magnate Charles Foster Kane, the movie does not proceed in chronological order through Kane's life. Instead, the film begins with Kane's death (famously murmuring "Rosebud") and is structured around flashbacks to his life inserted among scenes of a reporter investigating his death. The novel idea that the structure of a movie does not have to linearly follow the structure of the real timeline made apparent for 20th century cinematography the infinite possibilities and impact of different kinds of coherent narrative structures.

But coherent structure is not just a fact about movies or works of art. Like movies, language does not normally consist of isolated, unrelated sentences, but instead of collocated, structured, **coherent** groups of sentences. We refer to such a coherent structured group of sentences as a **discourse**, and we use the word **coherence** to refer to the relationship between sentences that makes real discourses different than just random assemblages of sentences. The chapter you are now reading is an example of a discourse, as is a news article, a conversation, a thread on social media, a Wikipedia page, and your favorite novel.

What makes a discourse coherent? If you created a text by taking a random sentences each from many different sources and pasted them together, would that be a coherent discourse? Almost certainly not. Real discourses exhibit both **local coherence** and **global coherence**. Let's consider three ways in which real discourses are locally coherent;

First, sentences or clauses in real discourses are related to nearby sentences in systematic ways. Consider this example from [Hobbs \(1979\)](#):

(23.1) John took a train from Paris to Istanbul. He likes spinach.

This sequence is incoherent because it is unclear to a reader why the second sentence follows the first; what does liking spinach have to do with train trips? In fact, a reader might go to some effort to try to figure out how the discourse could be coherent; perhaps there is a French spinach shortage? The very fact that hearers try to identify such connections suggests that human discourse comprehension involves the need to establish this kind of coherence.

By contrast, in the following coherent example:

(23.2) Jane took a train from Paris to Istanbul. She had to attend a conference.

discourse
coherence

local
global

coherence relations

the second sentence gives a REASON for Jane's action in the first sentence. Structured relationships like REASON that hold between text units are called coherence relations, and coherent discourses are structured by many such coherence relations. Coherence relations are introduced in Section 23.1.

A second way a discourse can be locally coherent is by virtue of being “about” someone or something. In a coherent discourse some entities are salient, and the discourse focuses on them and doesn't go back and forth between multiple entities. This is called entity-based coherence. Consider the following incoherent passage, in which the salient entity seems to wildly swing from John to Jenny to the piano store to the living room, back to Jenny, then the piano again:

- (23.3) John wanted to buy a piano for his living room.
 Jenny also wanted to buy a piano.
 He went to the piano store.
 It was nearby.
 The living room was on the second floor.
 She didn't find anything she liked.
 The piano he bought was hard to get up to that floor.

Centering Theory

Entity-based coherence models measure this kind of coherence by tracking salient entities across a discourse. For example Centering Theory (Grosz et al., 1995), the most influential theory of entity-based coherence, keeps track of which entities in the discourse model are salient at any point (salient entities are more likely to be pronominalized or to appear in prominent syntactic positions like subject or object). In Centering Theory, transitions between sentences that maintain the same salient entity are considered more coherent than ones that repeatedly shift between entities. The entity grid model of coherence (Barzilay and Lapata, 2008) is a commonly-used model that realizes some of the intuitions of the Centering Theory framework. Entity-based coherence is introduced in Section 23.3.

entity grid

Finally, discourses can be locally coherent by being topically coherent: nearby sentences are generally about the same topic and use the same or similar vocabulary to discuss these topics. Because topically coherent discourses draw from a single semantic field or topic, they tend to exhibit the surface property known as lexical cohesion (Halliday and Hasan, 1976): the sharing of identical or semantically related words in nearby sentences. For example, the fact that the words *house*, *chimney*, *garret*, *closet*, and *window*—all of which belong to the same semantic field—appear in the two sentences in (23.4), or that they share the identical word *shingled*, is a cue that the two are tied together as a discourse:

- (23.4) Before winter I built a chimney, and shingled the sides of my house...
 I have thus a tight shingled and plastered house... with a garret and a closet, a large window on each side....

lexical cohesion

In addition to the local coherence between adjacent or nearby sentences, discourses also exhibit global coherence. Many genres of text are associated with particular conventional discourse structures. Academic articles might have sections describing the Methodology or Results. Stories might follow conventional plotlines or motifs. Persuasive essays have a particular claim they are trying to argue for, and an essay might express this claim together with a structured set of premises that support the argument and demolish potential counterarguments. We'll introduce versions of each of these kinds of global coherence.

Why do we care about the local or global coherence of a discourse? Since coherence is a property of a well-written text, coherence detection plays a part in any

topically coherent

task that requires measuring the **quality** of a text. For example coherence can help in pedagogical tasks like essay grading or essay quality measurement that are trying to grade how well-written a human essay is (Somasundaran et al., 2014; Feng et al., 2014; Lai and Tetreault, 2018). Coherence can also help for summarization; knowing the coherence relationship between sentences can help know how to select information from them. Finally, detecting incoherent text may even play a role in mental health tasks like measuring symptoms of schizophrenia or other kinds of disordered language (Ditman and Kuperberg, 2010; Elvevåg et al., 2007; Bedi et al., 2015).

23.1 Coherence Relations

Recall from the introduction the difference between passages (23.5) and (23.6).

- (23.5) Jane took a train from Paris to Istanbul. She likes spinach.
- (23.6) Jane took a train from Paris to Istanbul. She had to attend a conference.

The reason (23.6) is more coherent is that the reader can form a connection between the two sentences, in which the second sentence provides a potential REASON for the first sentences. This link is harder to form for (23.5). These connections between text spans in a discourse can be specified as a set of **coherence relations**. The next two sections describe two commonly used models of coherence relations and associated corpora: Rhetorical Structure Theory (RST), and the Penn Discourse TreeBank (PDTB).

coherence relation

23.1.1 Rhetorical Structure Theory

RST
nucleus
satellite

The most commonly used model of discourse organization is **Rhetorical Structure Theory (RST)** (Mann and Thompson, 1987). In RST relations are defined between two spans of text, generally a **nucleus** and a **satellite**. The nucleus is the unit that is more central to the writer's purpose and that is interpretable independently; the satellite is less central and generally is only interpretable with respect to the nucleus. Some symmetric relations, however, hold between two nuclei.

Below are a few examples of RST coherence relations, with definitions adapted from the RST Treebank Manual (Carlson and Marcu, 2001).

Reason: The nucleus is an action carried out by an animate agent and the satellite is the reason for the nucleus.

- (23.7) [NUC Jane took a train from Paris to Istanbul.] [SAT She had to attend a conference.]

Elaboration: The satellite gives additional information or detail about the situation presented in the nucleus.

- (23.8) [NUC Dorothy was from Kansas.] [SAT She lived in the midst of the great Kansas prairies.]

Evidence: The satellite gives additional information or detail about the situation presented in the nucleus. The information is presented with the goal of convince the reader to accept the information presented in the nucleus.

- (23.9) [NUC Kevin must be here.] [SAT His car is parked outside.]

Attribution: The satellite gives the source of attribution for an instance of reported speech in the nucleus.

(23.10) [SAT Analysts estimated] [NUC that sales at U.S. stores declined in the quarter, too]

List: In this multinuclear relation, a series of nuclei is given, without contrast or explicit comparison:

(23.11) [NUC Billy Bones was the mate;] [NUC Long John, he was quartermaster]

RST relations are traditionally represented graphically; the asymmetric Nucleus-Satellite relation is represented with an arrow from the satellite to the nucleus:



We can also talk about the coherence of a larger text by considering the hierarchical structure between coherence relations. Figure 23.1 shows the rhetorical structure of a paragraph from Marcu (2000a) for the text in (23.12) from the *Scientific American* magazine.

(23.12) With its distant orbit—50 percent farther from the sun than Earth—and slim atmospheric blanket, Mars experiences frigid weather conditions. Surface temperatures typically average about -60 degrees Celsius (-76 degrees Fahrenheit) at the equator and can dip to -123 degrees C near the poles. Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion, but any liquid water formed in this way would evaporate almost instantly because of the low atmospheric pressure.



Figure 23.1 A discourse tree for the *Scientific American* text in (23.12), from Marcu (2000a). Note that asymmetric relations are represented with a curved arrow from the satellite to the nucleus.

The leaves in the Fig. 23.1 tree correspond to text spans of a sentence, clause or phrase that are called **elementary discourse units** or **EDUs** in RST; these units can also be referred to as **discourse segments**. Because these units may correspond to arbitrary spans of text, determining the boundaries of an EDU is an important task

for extracting coherence relations. Roughly speaking, one can think of discourse segments as being analogous to constituents in sentence syntax, and indeed as we'll see in Section 23.2 we generally draw on parsing algorithms to infer discourse structure.

There are corpora for many discourse coherence models; the RST Discourse TreeBank (Carlson et al., 2001) is the largest available discourse corpus. It consists of 385 English language documents selected from the Penn Treebank, with full RST parses for each one, using a large set of 78 distinct relations, grouped into 16 classes. RST treebanks exist also for Spanish, German, Basque, Dutch and Brazilian Portuguese (Braud et al., 2017).

Now that we've seen examples of coherence, we can see more clearly how a coherence relation can play a role in summarization or information extraction. For example, the nuclei of a text presumably express more important information than the satellites, which might be dropped in a summary.

23.1.2 Penn Discourse TreeBank (PDTB)

PDTB

discourse connectives

The **Penn Discourse TreeBank (PDTB)** is a second commonly used dataset that embodies another model of coherence relations (Miltsakaki et al., 2004; Prasad et al., 2008, 2014). PDTB labeling is *lexically grounded*. Instead of asking annotators to directly tag the coherence relation between text spans, they were given a list of **discourse connectives**, words that signal discourse relations, like *because*, *although*, *when*, *since*, or *as a result*. In a part of a text where these words marked a coherence relation between two text spans, the connective and the spans were then annotated, as in Fig. 23.13, where the phrase as a result signals a causal relationship between what PDTB calls *Arg1* (the first two sentences, here in italics) and **Arg2** (the third sentence, here in bold).

- (23.13) *Jewelry displays in department stores were often cluttered and uninspired. And the merchandise was, well, fake. **As a result**, marketers of faux gems steadily lost space in department stores to more fashionable rivals—cosmetics makers.*
- (23.14) *In July, the Environmental Protection Agency imposed a gradual ban on virtually all uses of asbestos. (implicit=as a result) **By 1997, almost all remaining uses of cancer-causing asbestos will be outlawed.***

Not all coherence relations are marked by an explicit discourse connective, and so the PDTB also annotates pairs of neighboring sentences with no explicit signal, like (23.14). The annotator first chooses the word or phrase that could have been its signal (in this case **as a result**), and then labels its sense. For example for the ambiguous discourse connective *since* annotators marked whether it is using a **CAUSAL** or a **TEMPORAL** sense.

The final dataset contains roughly 18,000 explicit relations and 16,000 implicit relations. Fig. 23.2 shows examples from each of the 4 major semantic classes, while Fig. 23.3 shows the full tagset.

Unlike the RST Discourse Treebank, which integrates these pairwise coherence relations into a global tree structure spanning an entire discourse, the PDTB does not annotate anything above the span-pair level, making no commitment with respect to higher-level discourse structure.

There are also treebanks using similar methods for other languages; (23.15) shows an example from the Chinese Discourse TreeBank (Zhou and Xue, 2015). Because Chinese has a smaller percentage of explicit discourse connectives than

Class	Type	Example
TEMPORAL	SYNCHRONOUS	The parishioners of St. Michael and All Angels stop to chat at the church door, as members here always have. (Implicit <u>while</u>) In the tower, five men and women pull rhythmically on ropes attached to the same five bells that first sounded here in 1614.
CONTINGENCY	REASON	Also unlike Mr. Ruder, Mr. Breeden appears to be in a position to get somewhere with his agenda. (implicit= <u>because</u>) As a former White House aide who worked closely with Congress, he is savvy in the ways of Washington.
COMPARISON	CONTRAST	The U.S. wants the removal of what it perceives as barriers to investment; Japan denies there are real barriers.
EXPANSION	CONJUNCTION	Not only do the actors stand outside their characters and make it clear they are at odds with them, <u>but</u> they often literally stand on their heads.

Figure 23.2 The four high-level semantic distinctions in the PDTB sense hierarchy

Temporal	Comparison
• Asynchronous	• Contrast (Juxtaposition, Opposition)
• Synchronous (Precedence, Succession)	• <i>Pragmatic Contrast (Juxtaposition, Opposition)</i>
	• Concession (Expectation, Contra-expectation)
	• <i>Pragmatic Concession</i>
Contingency	Expansion
• Cause (Reason, Result)	• <i>Exception</i>
• Pragmatic Cause (Justification)	• Instantiation
• <i>Condition (Hypothetical, General, Unreal Present/Past, Factual Present/Past)</i>	• Restatement (Specification, Equivalence, Generalization)
• <i>Pragmatic Condition (Relevance, Implicit Assertion)</i>	• Alternative (Conjunction, Disjunction, Chosen Alternative)
	• List

Figure 23.3 The PDTB sense hierarchy. There are four top-level classes, 16 types, and 23 subtypes (not all types have subtypes). 11 of the 16 types are commonly used for implicit argument classification; the 5 types in italics are too rare in implicit labeling to be used.

English (only 22% of all discourse relations are marked with explicit connectives, compared to 47% in English), annotators labeled this corpus by directly mapping pairs of sentences to 11 sense tags, without starting with a lexical discourse connector.

(23.15) [Conn 为] [Arg2 推动图们江地区开发] , [Arg1 韩国捐款一万美元设立了图们江发展基金]
“[In order to] [Arg2 promote the development of the Tumen River region], [Arg1 South Korea donated one million dollars to establish the Tumen River Development Fund].”

These discourse treebanks have been used for shared tasks on multilingual discourse parsing (Xue et al., 2016).

23.2 Discourse Structure Parsing

discourse
parsing

Given a sequence of sentences, how can we automatically determine the coherence relations between them? This task is often called **discourse parsing** (even though

for PDTB we are only assigning labels to leaf spans and not building a full parse tree as we do for RST).

23.2.1 EDU segmentation for RST parsing

RST parsing is generally done in two stages. The first stage, **EDU segmentation**, extracts the start and end of each EDU. The output of this stage would be a labeling like the following:

- (23.16) [Mr. Rambo says]_{e1} [that a 3.2-acre property]_{e2} [overlooking the San Fernando Valley]_{e3} [is priced at \$4 million]_{e4} [because the late actor Erroll Flynn once lived there]._{e5}

Since EDUs roughly correspond to clauses, early models of EDU segmentation first ran a syntactic parser, and then post-processed the output. Modern systems generally use neural sequence models supervised by the gold EDU segmentation in the RST Discourse Treebank. Fig. 23.4 shows an example from Wang et al. (2018) of a supervised architecture that uses the same biLSTM-CRF architecture we saw for named entity tagging and semantic role labeling. Here the input sentence is mapped to contextual word embeddings, and then passed through a biLSTM with a CRF layer on top to produce a sequence of 0s and 1s, where 1 indicates the start of an EDU (except at the start of sentence). Muller et al. (2019) find using BERT contextual embeddings plus convolutional character embeddings as the input to a similar biLSTM architecture produces highly accurate segmentations.

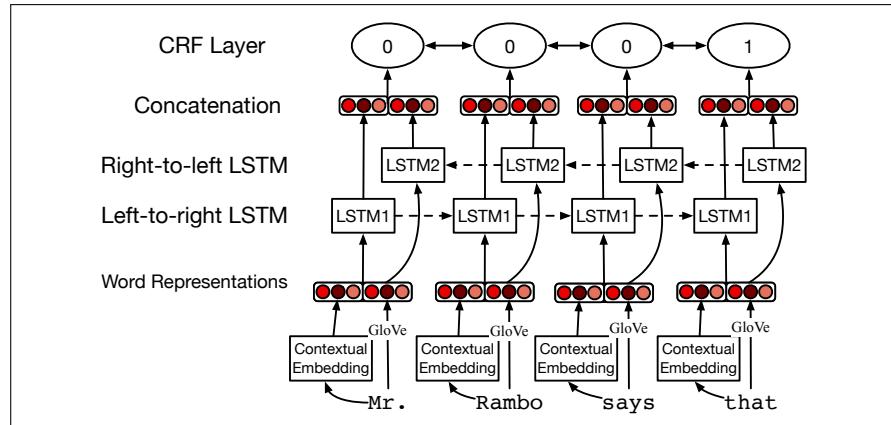


Figure 23.4 biLSTM-CRF for EDU segmentation. Word inputs can draw from any contextual embeddings like ELMo or BERT. After Wang et al. (2018).

23.2.2 RST parsing

Tools for building RST coherence structure for a discourse have long been based on syntactic parsing algorithms like shift-reduce parsing (Marcu, 1999). Many modern RST parsers since Ji and Eisenstein (2014) draw on the neural syntactic parsers we saw in Chapter 15 in Section 15.4, using representation learning to build representations for each span, and training a parser to choose the correct shift and reduce actions based on the gold parses in the training set.

We'll describe the shift-reduce parser of Yu et al. (2018). The parser state consists of a stack and a queue, and produces this structure by taking a series of actions on the states. Actions include:

- **shift**: pushes the first EDU in the queue onto the stack creating a single-node subtree.
- **reduce(l,d)**: merges the top two subtrees on the stack, where l is the coherence relation label, and d is the nuclearity direction, $d \in \{NN, NS, SN\}$.

As well as the **pop root** operation, to remove the final tree from the stack.



Figure 23.5 Example RST discourse tree, showing four EDUs. Figure from Yu et al. (2018).

Fig. 23.6 shows the actions the parser takes to build the structure in Fig. 23.5.

Step	Stack	Queue	Action	Relation
1	\emptyset	e_1, e_2, e_3, e_4	SH	\emptyset
2	e_1	e_2, e_3, e_4	SH	\emptyset
3	e_1, e_2	e_3, e_4	RD (attr, SN)	\emptyset
4	$e_{1:2}$	e_3, e_4	SH	$\widehat{e_1e_2}$
5	$e_{1:2}, e_3$	e_4	SH	$\widehat{e_1e_2}$
6	$e_{1:2}, e_3, e_4$	\emptyset	RD (elab, NS)	$\widehat{e_1e_2}$
7	$e_{1:2}, e_{3:4}$	\emptyset	RD (elab, SN)	$\widehat{e_1e_2}, \widehat{e_3e_4}$
8	$e_{1:4}$	\emptyset	PR	$\widehat{e_1e_2}, \widehat{e_3e_4}, \widehat{e_{1:2}e_{3:4}}$

Figure 23.6 Parsing the example of Fig. 23.5 using a shift-reduce parser. Figure from Yu et al. (2018).

The Yu et al. (2018) uses an encoder-decoder architecture, where the encoder represents the input span of words and EDUs using a hierarchical biLSTM. The first biLSTM layer represents the words inside an EDU, and the second represents the EDU sequence. Given an input sentence w_1, w_2, \dots, w_m , the words can be represented as usual (by static embeddings, combinations with character embeddings or tags, or contextual embeddings) resulting in an input word representation sequence $\mathbf{x}_1^w, \mathbf{x}_2^w, \dots, \mathbf{x}_m^w$. The result of the word-level biLSTM is then a sequence of \mathbf{h}^w values:

$$\mathbf{h}_1^w, \mathbf{h}_2^w, \dots, \mathbf{h}_m^w = \text{biLSTM}(\mathbf{x}_1^w, \mathbf{x}_2^w, \dots, \mathbf{x}_m^w) \quad (23.17)$$

An EDU of span w_s, w_{s+1}, \dots, w_t then has biLSTM output representation $\mathbf{h}_s^w, \mathbf{h}_{s+1}^w, \dots, \mathbf{h}_t^w$, and is represented by average pooling:

$$\mathbf{x}^e = \frac{1}{t-s+1} \sum_{k=s}^t \mathbf{h}_k^w \quad (23.18)$$

The second layer uses this input to compute a final representation of the sequence of EDU representations \mathbf{h}^e :

$$\mathbf{h}_1^e, \mathbf{h}_2^e, \dots, \mathbf{h}_n^e = \text{biLSTM}(\mathbf{x}_1^e, \mathbf{x}_2^e, \dots, \mathbf{x}_n^e) \quad (23.19)$$

The decoder is then a feedforward network \mathbf{W} that outputs an action o based on a concatenation of the top three subtrees on the stack (s_o, s_1, s_2) plus the first EDU in

the queue (q_0):

$$\mathbf{o} = \mathbf{W}(\mathbf{h}_{s0}^t, \mathbf{h}_{s0}^t, \mathbf{h}_{s1}^t, \mathbf{h}_{s2}^t, \mathbf{h}_{q0}^e) \quad (23.20)$$

where the representation of the EDU on the queue \mathbf{h}_{q0}^e comes directly from the encoder, and the three hidden vectors representing partial trees are computed by average pooling over the encoder output for the EDUs in those trees:

$$\mathbf{h}_s^t = \frac{1}{j-i+1} \sum_{k=i}^j \mathbf{h}_k^e \quad (23.21)$$

Training first maps each RST gold parse tree into a sequence of oracle actions, and then uses the standard cross-entropy loss (with l_2 regularization) to train the system to take such actions. Give a state S and oracle action a , we first compute the decoder output using Eq. 23.20, apply a softmax to get probabilities:

$$p_a = \frac{\exp(\mathbf{o}_a)}{\sum_{a' \in A} \exp(\mathbf{o}_{a'})} \quad (23.22)$$

and then computing the cross-entropy loss:

$$L(\Theta) = -\log(p_a) + \frac{\lambda}{2} \|\Theta\|^2 \quad (23.23)$$

RST discourse parsers are evaluated on the test section of the RST Discourse Treebank, either with gold EDUs or end-to-end, using the RST-Pareval metrics (Marcu, 2000b). It is standard to first transform the gold RST trees into right-branching binary trees, and to report four metrics: trees with no labels (S for Span), labeled with nuclei (N), with relations (R), or both (F for Full), for each metric computing micro-averaged F_1 over all spans from all documents (Marcu, 2000b; Morey et al., 2017).

23.2.3 PDTB discourse parsing

shallow
discourse
parsing

PDTB discourse parsing, the task of detecting PDTB coherence relations between spans, is sometimes called **shallow discourse parsing** because the task just involves flat relationships between text spans, rather than the full trees of RST parsing.

The set of four subtasks for PDTB discourse parsing was laid out by Lin et al. (2014) in the first complete system, with separate tasks for explicit (tasks 1-3) and implicit (task 4) connectives:

1. Find the discourse connectives (disambiguating them from non-discourse uses)
2. Find the two spans for each connective
3. Label the relationship between these spans
4. Assign a relation between every adjacent pair of sentences

Many systems have been proposed for Task 4: taking a pair of adjacent sentences as input and assign a coherence relation sense label as output. The setup often follows Lin et al. (2009) in assuming gold sentence span boundaries and assigning each adjacent span one of the 11 second-level PDTB tags or none (removing the 5 very rare tags of the 16 shown in italics in Fig. 23.3).

A simple but very strong algorithm for Task 4 is to represent each of the two spans by BERT contextual embeddings and take the last layer hidden state corresponding to the position of the <CLS> token, pass this through a single layer tanh feedforward network and then a softmax for sense classification (Nie et al., 2019).

Each of the other tasks also have been addressed. Task 1 is to disambiguating discourse connectives from their non-discourse use. For example as Pitler and Nenkova (2009) point out, the word *and* is a discourse connective linking the two clauses by an elaboration/expansion relation in (23.24) while it's a non-discourse NP conjunction in (23.25):

- (23.24) Selling picked up as previous buyers bailed out of their positions and aggressive short sellers—anticipating further declines—moved in.
- (23.25) My favorite colors are blue and green.

Similarly, *once* is a discourse connective indicating a temporal relation in (23.26), but simply a non-discourse adverb meaning ‘formerly’ and modifying *used* in (23.27):

- (23.26) The asbestos fiber, crocidolite, is unusually resilient once it enters the lungs, with even brief exposures to it causing symptoms that show up decades later, researchers said.
- (23.27) A form of asbestos once used to make Kent cigarette filters has caused a high percentage of cancer deaths among a group of workers exposed to it more than 30 years ago, researchers reported.

Determining whether a word is a discourse connective is thus a special case of word sense disambiguation. Early work on disambiguation showed that the 4 PDTB high-level sense classes could be disambiguated with high (94%) accuracy used syntactic features from gold parse trees (Pitler and Nenkova, 2009). Recent work performs the task end to end from word inputs using a biLSTM-CRF with BIO outputs (B-CONN, I-CONN, O) (Yu et al., 2019).

For task 2, PDTB spans can be identified with the same sequence models used to find RST EDUs: a biLSTM sequence model with pretrained contextual embedding (BERT) inputs (Muller et al., 2019). Simple heuristics also do pretty well as a baseline at finding spans, since 93% of relations are either completely within a single sentence or span two adjacent sentences, with one argument in each sentence (Biran and McKeown, 2015).

23.3 Centering and Entity-Based Coherence

entity-based

A second way a discourse can be coherent is by virtue of being “about” some entity. This idea that at each point in the discourse some entity is salient, and a discourse is coherent by continuing to discuss the same entity, appears early in functional linguistics and the psychology of discourse (Chafe, 1976; Kintsch and Van Dijk, 1978), and soon made its way to computational models. In this section we introduce two models of this kind of **entity-based coherence**: **Centering Theory** (Grosz et al., 1995), and the **entity grid** model of Barzilay and Lapata (2008).

23.3.1 Centering

Centering Theory

Centering Theory (Grosz et al., 1995) is a theory of both discourse salience and discourse coherence. As a model of discourse salience, Centering proposes that at any given point in the discourse one of the entities in the discourse model is salient: it is being “centered” on. As a model of discourse coherence, Centering proposes that discourses in which adjacent sentences CONTINUE to maintain the same salient entity are more coherent than those which SHIFT back and forth between multiple entities (we will see that CONTINUE and SHIFT are technical terms in the theory).

The following two texts from [Grosz et al. \(1995\)](#) which have exactly the same propositional content but different saliences, can help in understanding the main Centering intuition.

- (23.28) 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.

- (23.29) 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.

While these two texts differ only in how the two entities (John and the store) are realized in the sentences, the discourse in (23.28) is intuitively more coherent than the one in (23.29). As [Grosz et al. \(1995\)](#) point out, this is because the discourse in (23.28) is clearly about one individual, John, describing his actions and feelings. The discourse in (23.29), by contrast, focuses first on John, then the store, then back to John, then to the store again. It lacks the “aboutness” of the first discourse.

backward-looking center
forward-looking center

Centering Theory realizes this intuition by maintaining two representations for each utterance U_n . The **backward-looking center** of U_n , denoted as $C_b(U_n)$, represents the current salient entity, the one being focused on in the discourse after U_n is interpreted. The **forward-looking centers** of U_n , denoted as $C_f(U_n)$, are a set of potential future salient entities, the discourse entities evoked by U_n any of which could serve as C_b (the salient entity) of the following utterance, i.e. $C_b(U_{n+1})$.

The set of forward-looking centers $C_f(U_n)$ are ranked according to factors like discourse salience and grammatical role (for example subjects are higher ranked than objects, which are higher ranked than all other grammatical roles). We call the highest-ranked forward-looking center C_p (for “preferred center”). C_p is a kind of prediction about what entity will be talked about next. Sometimes the next utterance indeed talks about this entity, but sometimes another entity becomes salient instead.

We'll use here the algorithm for centering presented in [Brennan et al. \(1987\)](#), which defines four intersentential relationships between a pair of utterances U_n and U_{n+1} that depend on the relationship between $C_b(U_{n+1})$, $C_b(U_n)$, and $C_p(U_{n+1})$; these are shown in Fig. 23.7.

$C_b(U_{n+1}) = C_b(U_n)$ or undefined $C_b(U_n)$	$C_b(U_{n+1}) \neq C_b(U_n)$	
$C_b(U_{n+1}) = C_p(U_{n+1})$	Continue	Smooth-Shift
$C_b(U_{n+1}) \neq C_p(U_{n+1})$	Retain	Rough-Shift

Figure 23.7 Centering Transitions for Rule 2 from [Brennan et al. \(1987\)](#).

The following rules are used by the algorithm:

- Rule 1:** If any element of $C_f(U_n)$ is realized by a pronoun in utterance U_{n+1} , then $C_b(U_{n+1})$ must be realized as a pronoun also.
Rule 2: Transition states are ordered. Continue is preferred to Retain is preferred to Smooth-Shift is preferred to Rough-Shift.

Rule 1 captures the intuition that pronominalization (including zero-anaphora) is a common way to mark discourse salience. If there are multiple pronouns in an

utterance realizing entities from the previous utterance, one of these pronouns must realize the backward center C_b ; if there is only one pronoun, it must be C_b .

Rule 2 captures the intuition that discourses that continue to center the same entity are more coherent than ones that repeatedly shift to other centers. The transition table is based on two factors: whether the backward-looking center C_b is the same from U_n to U_{n+1} and whether this discourse entity is the one that was preferred (C_p) from U_n . If both of these hold, a CONTINUE relation, the speaker has been talking about the same entity and is going to continue talking about that entity. In a RETAIN relation, the speaker intends to SHIFT to a new entity in a future utterance and meanwhile places the current entity in a lower rank C_f . In a SHIFT relation, the speaker is shifting to a new salient entity.

Let's walk though the start of (23.28) again, repeated as (23.30), showing the representations after each utterance is processed.

- (23.30) John went to his favorite music store to buy a piano. (U_1)
 - He was excited that he could finally buy a piano. (U_2)
 - He arrived just as the store was closing for the day. (U_3)
 - It was closing just as John arrived (U_4)

Using the grammatical role hierarchy to order the C_f , for sentence U_1 we get:

- $C_f(U_1)$: {John, music store, piano}
- $C_p(U_1)$: John
- $C_b(U_1)$: undefined

and then for sentence U_2 :

- $C_f(U_2)$: {John, piano}
 - $C_p(U_2)$: John
 - $C_b(U_2)$: John
- Result: Continue ($C_p(U_2)=C_b(U_2)$; $C_b(U_1)$ undefined)

The transition from U_1 to U_2 is thus a CONTINUE. Completing this example is left as exercise (1) for the reader

23.3.2 Entity Grid model

Centering embodies a particular theory of how entity mentioning leads to coherence: that salient entities appear in subject position or are pronominalized, and that discourses are salient by means of continuing to mention the same entity in such ways.

entity grid

The **entity grid** model of Barzilay and Lapata (2008) is an alternative way to capture entity-based coherence: instead of having a top-down theory, the entity-grid model uses machine learning to induce the patterns of entity mentioning that make a discourse more coherent.

The model is based around an **entity grid**, a two-dimensional array that represents the distribution of entity mentions across sentences. The rows represent sentences, and the columns represent discourse entities (most versions of the entity grid model focus just on nominal mentions). Each cell represents the possible appearance of an entity in a sentence, and the values represent whether the entity appears and its grammatical role. Grammatical roles are subject (s), object (o), neither (x), or absent (–); in the implementation of Barzilay and Lapata (2008), subjects of passives are represented with o, leading to a representation with some of the characteristics of thematic roles.

	Department	Trial	Microsoft	Evidence	Competitors	Markets	Products	Brands	Case	Netscape	Software	Tactics	Govern	Suit	Earnings
1	s	o	s	x	o	-	-	-	-	-	-	-	-	-	1
2	-	-	o	-	-	x	s	o	-	-	-	-	-	-	2
3	-	-	s	o	-	-	-	s	o	o	-	-	-	-	3
4	-	-	s	-	-	-	-	-	-	s	-	-	-	-	4
5	-	-	-	-	-	-	-	-	-	-	s	o	-	-	5
6	-	x	s	-	-	-	-	-	-	-	-	-	o	6	

Figure 23.8 Part of the entity grid for the text in Fig. 23.9. Entities are listed by their head noun; each cell represents whether an entity appears as subject (S), object (O), neither (X), or is absent (-). Figure from Barzilay and Lapata (2008).

1	[The Justice Department] _s is conducting an [anti-trust trial] _o against [Microsoft Corp.] _x with [evidence] _x that [the company] _s is increasingly attempting to crush [competitors] _o .
2	[Microsoft] _o is accused of trying to forcefully buy into [markets] _x where [its own products] _s are not competitive enough to unseat [established brands] _o .
3	[The case] _s revolves around [evidence] _o of [Microsoft] _s aggressively pressuring [Netscape] _o into merging [browser software] _o .
4	[Microsoft] _s claims [its tactics] _s are commonplace and good economically.
5	[The government] _s may file [a civil suit] _o ruling that [conspiracy] _s to curb [competition] _o through [collusion] _x is [a violation of the Sherman Act] _o .
6	[Microsoft] _s continues to show [increased earnings] _o despite [the trial] _x .

Figure 23.9 A discourse with the entities marked and annotated with grammatical functions. Figure from Barzilay and Lapata (2008).

Fig. 23.8 from Barzilay and Lapata (2008) shows a grid for the text shown in Fig. 23.9. There is one row for each of the six sentences. The second column, for the entity ‘trial’, is O – – – X, showing that the trial appears in the first sentence as direct object, in the last sentence as an oblique, and does not appear in the middle sentences. The third column, for the entity Microsoft, shows that it appears as subject in sentence 1 (it also appears as the object of the preposition *against*, but entities that appear multiple times are recorded with their highest-ranked grammatical function). Computing the entity grids requires extracting entities and doing coreference resolution to cluster them into discourse entities (Chapter 22) as well as parsing the sentences to get grammatical roles.

In the resulting grid, columns that are dense (like the column for Microsoft) indicate entities that are mentioned often in the texts; sparse columns (like the column for earnings) indicate entities that are mentioned rarely.

In the entity grid model, coherence is measured by patterns of **local entity transition**. For example, Department is a subject in sentence 1, and then not mentioned in sentence 2; this is the transition [S –]. The transitions are thus sequences {S,O,X,–}ⁿ which can be extracted as continuous cells from each column. Each transition has a probability; the probability of [S –] in the grid from Fig. 23.8 is 0.08 (it occurs 6 times out of the 75 total transitions of length two). Fig. 23.10 shows the distribution over transitions of length 2 for the text of Fig. 23.9 (shown as the first row d_1), and 2 other documents.

The transitions and their probabilities can then be used as features for a machine learning model. This model can be a text classifier trained to produce human-labeled coherence scores (for example from humans labeling each text as coherent or incoherent). But such data is expensive to gather. Barzilay and Lapata (2005) introduced a simplifying innovation: coherence models can be trained by **self-supervision**: trained to distinguish the natural original order of sentences in a discourse from

	s s	s o	s x	s -	o s	o o	o x	o -	x s	x o	x x	x -	- s	- o	- x	--
d_1	.01	.01	0	.08	.01	0	0	.09	0	0	0	.03	.05	.07	.03	.59
d_2	.02	.01	.01	.02	0	.07	0	.02	.14	.14	.06	.04	.03	.07	0.1	.36
d_3	.02	0	0	.03	.09	0	.09	.06	0	0	.05	.03	.07	.17	.39	

Figure 23.10 A feature vector for representing documents using all transitions of length 2. Document d_1 is the text in Fig. 23.9. Figure from [Barzilay and Lapata \(2008\)](#).

a modified order (such as a randomized order). We turn to these evaluations in the next section.

23.3.3 Evaluating Neural and Entity-based coherence

Entity-based coherence models, as well as the neural models we introduce in the next section, are generally evaluated in one of two ways.

First, we can have humans rate the coherence of a document and train a classifier to predict these human ratings, which can be categorial (high/low, or high/mid/low) or continuous. This is the best evaluation to use if we have some end task in mind, like essay grading, where human raters are the correct definition of the final label.

Alternatively, since it's very expensive to get human labels, and we might not yet have an end-task in mind, we can use natural texts to do self-supervision. In self-supervision we pair up a natural discourse with a pseudo-document created by changing the ordering. Since naturally-ordered discourses are more coherent than random permutation ([Lin et al., 2011](#)), a successful coherence algorithm should prefer the original ordering.

Self-supervision has been implemented in 3 ways. In the **sentence order discrimination** task ([Barzilay and Lapata, 2005](#)), we compare a document to a random permutation of its sentence. A model is considered correct for an (original, permuted) test pair if it ranks the original document higher. Given k documents, we can compute n permutations, resulting in kn pairs each with one original document and one permutation, to use in training and testing.

In the **sentence insertion** task ([Chen et al., 2007](#)) we take a document, remove one of the n sentences s , and create $n - 1$ copies of the document with s inserted into each position. The task is to decide which of the n documents is the one with the original ordering, distinguishing the original position for s from all other positions. Insertion is harder than discrimination since we are comparing documents that differ by only one sentence.

Finally, in the **sentence order reconstruction** task ([Lapata, 2003](#)), we take a document, randomize the sentences, and train the model to put them back in the correct order. Again given k documents, we can compute n permutations, resulting in kn pairs each with one original document and one permutation, to use in training and testing. Reordering is of course a much harder task than simple classification.

23.4 Representation learning models for local coherence

The third kind of local coherence is topical or semantic field coherence. Discourses cohere by talking about the same topics and subtopics, and drawing on the same semantic fields in doing so.

The field was pioneered by a series of unsupervised models in the 1990s of this

lexical cohesion	kind of coherence that made use of lexical cohesion (Halliday and Hasan, 1976): the sharing of identical or semantically related words in nearby sentences. Morris and Hirst (1991) computed lexical chains of words (like <i>pine, bush trees, trunk</i>) that occurred through a discourse and that were related in Roget's Thesaurus (by being in the same category, or linked categories). They showed that the number and density of chain correlated with the topic structure. The TextTiling algorithm of Hearst (1997) computed the cosine between neighboring text spans (the normalized dot product of vectors of raw word counts), again showing that sentences or paragraph in a subtopic have high cosine with each other, but not with sentences in a neighboring subtopic.
TextTiling	A third early model, the LSA Coherence method of Foltz et al. (1998) was the first to use embeddings, modeling the coherence between two sentences as the cosine between their LSA sentence embedding vectors ¹ , computing embeddings for a sentence s by summing the embeddings of its words w :

$$\begin{aligned} \text{sim}(s, t) &= \cos(\mathbf{s}, \mathbf{t}) \\ &= \cos\left(\sum_{w \in s} \mathbf{w}, \sum_{w \in t} \mathbf{w}\right) \end{aligned} \quad (23.31)$$

and defining the overall coherence of a text as the average similarity over all pairs of adjacent sentences s_i and s_{i+1} :

$$\text{coherence}(T) = \frac{1}{n-1} \sum_{i=1}^{n-1} \cos(s_i, s_{i+1}) \quad (23.32)$$

Modern neural representation-learning coherence models, beginning with Li et al. (2014), draw on the intuitions of these early unsupervised models for learning sentence representations and measuring how they change between neighboring sentences. But the new models also draw on the idea pioneered by Barzilay and Lapata (2005) of self-supervision. That is, unlike say coherence relation models, which train on hand-labeled representations for RST or PDTB, these models are trained to distinguish natural discourses from unnatural discourses formed by scrambling the order of sentences, thus using representation learning to discover the features that matter for at least the ordering aspect of coherence.

Here we present one such model, the local coherence discriminator (LCD) (Xu et al., 2019). Like early models, LCD computes the coherence of a text as the average of coherence scores between consecutive pairs of sentences. But unlike the early unsupervised models, LCD is a self-supervised model trained to discriminate consecutive sentence pairs (s_i, s_{i+1}) in the training documents (assumed to be coherent) from (constructed) incoherent pairs (s_i, s') . All consecutive pairs are positive examples, and the negative (incoherent) partner for a sentence s_i is another sentence uniformly sampled from the same document as s_i .

Fig. 23.11 describes the architecture of the model f_θ , which takes a sentence pair and returns a score, higher scores for more coherent pairs. Given an input sentence pair s and t , the model computes sentence embeddings \mathbf{s} and \mathbf{t} (using any sentence embeddings algorithm), and then concatenates four features of the pair: (1) the concatenation of the two vectors (2) their difference $\mathbf{s} - \mathbf{t}$; (3) the absolute value of their difference $|\mathbf{s} - \mathbf{t}|$; (4) their element-wise product $\mathbf{s} \odot \mathbf{t}$. These are passed through a one-layer feedforward network to output the coherence score.

¹ See Chapter 6 for more on LSA embeddings; they are computed by applying SVD to the term-document matrix (each cell weighted by log frequency and normalized by entropy), and then the first 300 dimensions are used as the embedding.

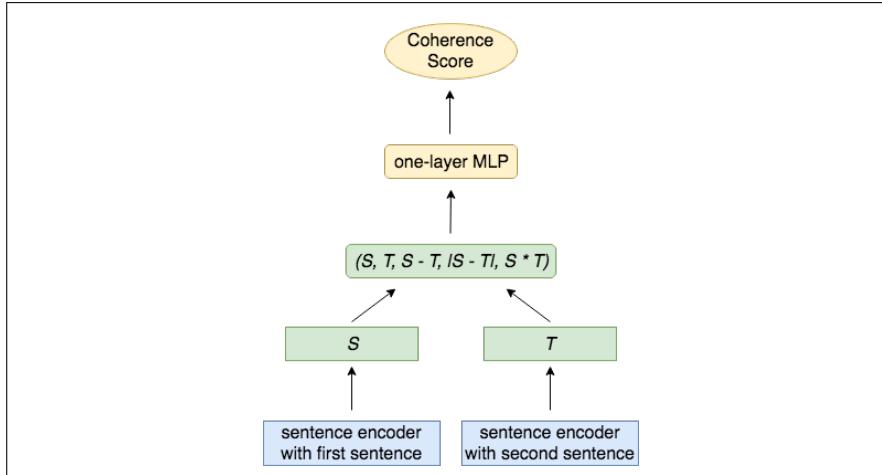


Figure 23.11 The architecture of the LCD model of document coherence, showing the computation of the score for a pair of sentences s and t . Figure from [Xu et al. \(2019\)](#).

The model is trained to make this coherence score higher for real pairs than for negative pairs. More formally, the training objective for a corpus C of documents d , each of which consists of a list of sentences s_i , is:

$$L_\theta = \sum_{d \in C} \sum_{s_i \in d} \mathbb{E}_{p(s'|s_i)} [L(f_\theta(s_i, s_{i+1}), f_\theta(s_i, s'))] \quad (23.33)$$

$\mathbb{E}_{p(s'|s_i)}$ is the expectation with respect to the negative sampling distribution conditioned on s_i : given a sentence s_i the algorithm samples a negative sentence s' uniformly over the other sentences in the same document. L is a loss function that takes two scores, one for a positive pair and one for a negative pair, with the goal of encouraging $f^+ = f_\theta(s_i, s_{i+1})$ to be high and $f^- = f_\theta(s_i, s')$ to be low. Fig. 23.11 use the margin loss $l(f^+, f^-) = \max(0, \eta - f^+ + f^-)$ where η is the margin hyperparameter.

[Xu et al. \(2019\)](#) also give a useful baseline algorithm that itself has quite high performance in measuring perplexity: train an RNN language model on the data, and compute the log likelihood of sentence s_i in two ways, once given the preceding context (conditional log likelihood) and once with no context (marginal log likelihood). The difference between these values tells us how much the preceding context improved the predictability of s_i , a predictability measure of coherence.

While the architecture and roots of these neural models lie in the cohesion-based idea that coherent discourses share words, semantic fields, and topics, qualitative analysis of these models ([Li and Jurafsky, 2017](#)) suggest that neural models may also represent coherence due to relations (for example neural models find pairs of sentences coherent when they have a causal or temporal relation) and entity coherence (for example the models correctly assign discourse (23.28) a higher coherence score than (23.29)).

23.5 Global Coherence

A discourse must also cohere globally rather than just at the level of pairs of sentences. Consider stories, for example. The narrative structure of stories is one of

the oldest kinds of global coherence to be studied. In his influential *Morphology of the Folktale*, Propp (1968) models the discourse structure of Russian folktales via a kind of plot grammar. His model includes a set of character categories he called **dramatis personae**, like Hero, Villain, Donor, or Helper, and a set of events he called **functions** (like “Villain commits kidnapping”, “Donor tests Hero”, or “Hero is pursued”) that have to occur in particular order, along with other components. Propp shows that the plots of each of the fairy tales he studies can be represented as a sequence of these functions, different tales choosing different subsets of functions, but always in the same order. Indeed Lakoff (1972b) showed that Propp’s model amounted to a discourse grammar of stories, and in recent computational work Finlayson (2016) demonstrates that some of these Proppian functions could be induced from corpora of folktale texts by detecting events that have similar actions across stories. Bamman et al. (2013) showed that generalizations over dramatis personae could be induced from movie plot summaries on Wikipedia. Their model induced latent personae from features like the actions the character takes (e.g., Villains strangle), the actions done to them (e.g., Villains are foiled and arrested) or the descriptive words used of them (Villains are evil).

In this section we introduce two kinds of such global discourse structure that have been widely studied computationally. The first is the structure of arguments: the way people attempt to convince each other in persuasive essays by offering claims and supporting premises. The second is somewhat related: the structure of scientific papers, and the way authors present their goals, results, and relationship to prior work in their papers.

23.5.1 Argumentation Structure

argumentation mining

The first type of global discourse structure is the structure of **arguments**. Analyzing people’s argumentation computationally is often called **argumentation mining**.

pathos
ethos
logos

The study of arguments dates back to Aristotle, who in his Rhetorics described three components of a good argument: **pathos** (appealing to the emotions of the listener), **ethos** (appealing to the speaker’s personal character), and **logos** (the logical structure of the argument).

Most of the discourse structure studies of argumentation have focused on **logos**, particularly via building and training on annotated datasets of persuasive essays or other arguments (Reed et al., 2008; Stab and Gurevych, 2014a; Peldszus and Stede, 2016; Habernal and Gurevych, 2017; Musi et al., 2018). Such corpora, for example, often include annotations of argumentative components like **claims** (the central component of the argument that is controversial and needs support) and **premises** (the reasons given by the author to persuade the reader by supporting or attacking the claim or other premises), as well as the **argumentative relations** between them like SUPPORT and ATTACK.

Consider the following example of a persuasive essay from Stab and Gurevych (2014b). The first sentence (1) presents a claim (in bold). (2) and (3) present two premises supporting the claim. (4) gives a premise supporting premise (3).

claims
premises
argumentative relations

“(1) Museums and art galleries provide a better understanding about arts than Internet. (2) In most museums and art galleries, detailed descriptions in terms of the background, history and author are provided. (3) Seeing an artwork online is not the same as watching it with our own eyes, as (4) the picture online does not show the texture or three-dimensional structure of the art, which is important to study.”

Thus this example has three argumentative relations: SUPPORT(2,1), SUPPORT(3,1) and SUPPORT(4,3). Fig. 23.12 shows the structure of a much more complex argument.



Figure 23.12 Argumentation structure of a persuasive essay. Arrows indicate argumentation relations, either of SUPPORT (with arrowheads) or ATTACK (with circleheads); P denotes premises. Figure from [Stab and Gurevych \(2017\)](#).

While argumentation mining is clearly related to rhetorical structure and other kinds of coherence relations, arguments tend to be much less local; often a persuasive essay will have only a single main claim, with premises spread throughout the text, without the local coherence we see in coherence relations.

Algorithms for detecting argumentation structure often include classifiers for distinguishing claims, premises, or non-argumentation, together with relation classifiers for deciding if two spans have the SUPPORT, ATTACK, or neither relation ([Peldszus and Stede, 2013](#)). While these are the main focus of much computational work, there is also preliminary efforts on annotating and detecting richer semantic relationships ([Park and Cardie, 2014; Hidey et al., 2017](#)) such as detecting **argumentation schemes**, larger-scale structures for argument like **argument from example**, or **argument from cause to effect**, or **argument from consequences** ([Feng and Hirst, 2011](#)).

argumentation schemes

persuasion

Another important line of research is studying how these argument structure (or other features) are associated with the success or persuasiveness of an argument ([Habernal and Gurevych, 2016; Tan et al., 2016; Hidey et al., 2017](#)). Indeed, while it is Aristotle's logos that is most related to discourse structure, Aristotle's ethos and pathos techniques are particularly relevant in the detection of mechanisms of this sort of **persuasion**. For example scholars have investigated the linguistic realization of features studied by social scientists like **reciprocity** (people return favors), **social proof** (people follow others' choices), **authority** (people are influenced by those with power), and **scarcity** (people value things that are scarce), all of which can be brought up in a persuasive argument ([Cialdini, 1984](#)). [Rosenthal and McKeown \(2017\)](#) showed that these features could be combined with argumentation structure to predict who influences whom on social media, [Althoff et al. \(2014\)](#) found that linguistic models of reciprocity and authority predicted success in online requests, while the semisupervised model of [Yang et al. \(2019\)](#) detected mentions of scarcity, commitment, and social identity to predict the success of peer-to-peer lending plat-

forms.

See [Stede and Schneider \(2018\)](#) for a comprehensive survey of argument mining.

argumentative zoning

23.5.2 The structure of scientific discourse

Scientific papers have a very specific global structure: somewhere in the course of the paper the authors must indicate a scientific goal, develop a method for a solution, provide evidence for the solution, and compare to prior work. One popular annotation scheme for modeling these rhetorical goals is the **argumentative zoning** model of [Teufel et al. \(1999\)](#) and [Teufel et al. \(2009\)](#), which is informed by the idea that each scientific paper tries to make a **knowledge claim** about a new piece of knowledge being added to the repository of the field ([Myers, 1992](#)). Sentences in a scientific paper can be assigned one of 15 tags; Fig. 23.13 shows 7 (shortened) examples of labeled sentences.

Category	Description	Example
AIM	Statement of specific research goal, or hypothesis of current paper	“The aim of this process is to examine the role that training plays in the tagging process”
OWN_METHOD	New Knowledge claim, own work: methods	“In order for it to be useful for our purposes, the following extensions must be made:”
OWN_RESULTS	Measurable/objective outcome of own work	“All the curves have a generally upward trend but always lie far below backoff (51% error rate)”
USE	Other work is used in own work	“We use the framework for the allocation and transfer of control of Whittaker....”
GAP_WEAK	Lack of solution in field, problem with other solutions	“Here, we will produce experimental evidence suggesting that this simple model leads to serious overestimates”
SUPPORT	Other work supports current work or is supported by current work	“Work similar to that described here has been carried out by Merialdo (1994), with broadly similar conclusions.”
ANTISUPPORT	Clash with other’s results or theory; superiority of own work	“This result challenges the claims of...”

Figure 23.13 Examples for 7 of the 15 labels from the Argumentative Zoning labelset ([Teufel et al., 2009](#)).

[Teufel et al. \(1999\)](#) and [Teufel et al. \(2009\)](#) develop labeled corpora of scientific articles from computational linguistics and chemistry, which can be used as supervision for training standard sentence-classification architecture to assign the 15 labels.

23.6 Summary

In this chapter we introduced local and global models for discourse **coherence**.

- Discourses are not arbitrary collections of sentences; they must be *coherent*. Among the factors that make a discourse coherent are coherence relations between the sentences, entity-based coherence, and topical coherence.
- Various sets of **coherence relations** and **rhetorical relations** have been proposed. The relations in Rhetorical Structure Theory (**RST**) hold between spans of text and are structured into a tree. Because of this, shift-reduce and other parsing algorithms are generally used to assign these structures. The Penn Discourse Treebank (**PDTB**) labels only relations between pairs of spans, and the labels are generally assigned by sequence models.

- **Entity-based coherence** captures the intuition that discourses are **about** an entity, and continue mentioning the entity from sentence to sentence. **Centering Theory** is a family of models describing how salience is modeled for discourse entities, and hence how coherence is achieved by virtue of keeping the same discourse entities salient over the discourse. The **entity grid** model gives a more bottom-up way to compute which entity realization transitions lead to coherence.
- Many different genres have different types of **global coherence**. Persuasive essays have claims and premises that are extracted in the field of **argument mining**, scientific articles have structure related to aims, methods, results, and comparisons.

Bibliographical and Historical Notes

SDRT

Coherence relations arose from the independent development of a number of scholars, including [Hobbs \(1979\)](#) idea that coherence relations play an inferential role for the hearer, and the investigations by [Mann and Thompson \(1987\)](#) of the discourse structure of large texts. Other approaches to coherence relations and their extraction include Segmented Discourse Representation Theory (**SDRT**) ([Asher and Lascardes 2003](#), [Baldridge et al. 2007](#)) and the Linguistic Discourse Model ([Polanyi, 1988](#); [Scha and Polanyi, 1988](#); [Polanyi et al., 2004a, 2004b](#)) [Wolf and Gibson \(2005\)](#) argue that coherence structure includes crossed bracketings, which make it impossible to represent as a tree, and propose a graph representation instead. A compendium of over 350 relations that have been proposed in the literature can be found in [Hovy \(1990\)](#).

RST parsing was first proposed by [Marcu \(1997\)](#), and early work was rule-based, focused on discourse markers ([Marcu, 2000a](#)). The creation of the RST Discourse TreeBank ([Carlson et al. 2001](#), [Carlson and Marcu 2001](#)) enabled a wide variety of machine learning algorithms, beginning with the shift-reduce parser of [Marcu \(1999\)](#) that used decision trees to choose actions, and continuing with a wide variety of machine learned parsing methods ([Soricut and Marcu 2003](#), [Sagae 2009](#), [Hernault et al. 2010](#), [Feng and Hirst 2014](#), [Surdeanu et al. 2015](#), [Joty et al. 2015](#)) and chunkers ([Sporleder and Lapata, 2005](#)). [Subba and Di Eugenio \(2009\)](#) integrated sophisticated semantic information into RST parsing. [Ji and Eisenstein \(2014\)](#) first applied neural models to RST parsing neural models, leading to the modern set of neural RST models ([Li et al. 2014](#), [Li et al. 2016](#), [Braud et al. 2017](#), [Yu et al. 2018](#), *inter alia*) as well as neural segmenters ([Wang et al. 2018](#)), and neural PDTB parsing models ([Ji and Eisenstein 2015](#), [Qin et al. 2016](#), [Qin et al. 2017](#)).

[Barzilay and Lapata \(2005\)](#) pioneered the idea of self-supervision for coherence: training a coherence model to distinguish true orderings of sentences from random permutations. [Li et al. \(2014\)](#) first applied this paradigm to neural sentence-representation, and a many neural self-supervised models followed ([Li and Jurafsky 2017](#), [Logeswaran et al. 2018](#), [Lai and Tetreault 2018](#), [Xu et al. 2019](#))

Another aspect of global coherence is the global topic structure of a text, the way the topics shift over the course of the document. [Barzilay and Lee \(2004\)](#) introduced an HMM model for capturing topics for coherence, and later work expanded this intuition ([Soricut and Marcu 2006](#), [Elsner et al. 2007](#), [Louis and Nenkova 2012](#), [Li and Jurafsky 2017](#)).

The relationship between explicit and implicit discourse connectives has been a fruitful one for research. [Marcu and Echihabi \(2002\)](#) first proposed to use sentences with explicit relations to help provide training data for implicit relations, by removing the explicit relations and trying to re-predict them as a way of improving performance on implicit connectives; this idea was refined by [Sporleder and Lascarides \(2005\)](#), [\(Pitler et al., 2009\)](#), and [Rutherford and Xue \(2015\)](#). This relationship can also be used as a way to create discourse-aware representations. The DisSent algorithm ([Nie et al., 2019](#)) creates the task of predicting explicit discourse markers between two sentences. They show that representations learned to be good at this task also function as powerful sentence representations for other discourse tasks.

The idea of entity-based coherence seems to have arisen in multiple fields in the mid-1970s, in functional linguistics ([Chafe, 1976](#)), in the psychology of discourse processing ([Kintsch and Van Dijk, 1978](#)), and in the roughly contemporaneous work of Grosz, Sidner, Joshi, and their colleagues. [Grosz \(1977a\)](#) addressed the focus of attention that conversational participants maintain as the discourse unfolds. She defined two levels of focus; entities relevant to the entire discourse were said to be in *global* focus, whereas entities that are locally in focus (i.e., most central to a particular utterance) were said to be in *immediate* focus. [Sidner \(1979, 1983\)](#) described a method for tracking (immediate) discourse foci and their use in resolving pronouns and demonstrative noun phrases. She made a distinction between the current discourse focus and potential foci, which are the predecessors to the backward- and forward-looking centers of Centering theory, respectively. The name and further roots of the centering approach lie in papers by [Joshi and Kuhn \(1979\)](#) and [Joshi and Weinstein \(1981\)](#), who addressed the relationship between immediate focus and the inferences required to integrate the current utterance into the discourse model. [Grosz et al. \(1983\)](#) integrated this work with the prior work of Sidner and Grosz. This led to a manuscript on centering which, while widely circulated since 1986, remained unpublished until [Grosz et al. \(1995\)](#). A collection of centering papers appears in [Walker et al. \(1998\)](#). See [Karamanis et al. \(2004\)](#) and [Poesio et al. \(2004\)](#) for a deeper exploration of centering and its parameterizations, and the History section of Chapter 22 for more on the use of centering on coreference.

The grid model of entity-based coherence was first proposed by [Barzilay and Lapata \(2005\)](#) drawing on earlier work by [Lapata \(2003\)](#) and Barzilay, and then extended by them [Barzilay and Lapata \(2008\)](#) and others with additional features ([Elsner and Charniak 2008, \(2011\)](#), [Feng et al. 2014](#), [Lin et al. 2011](#)) a model that projects entities into a global graph for the discourse ([Guinaudeau and Strube 2013](#), [Mesgar and Strube 2016](#)), and a convolutional model to capture longer-range entity dependencies ([Nguyen and Joty, 2017](#)).

Theories of discourse coherence have also been used in algorithms for interpreting discourse-level linguistic phenomena, including verb phrase ellipsis and gapping ([Asher, 1993; Kehler, 1993](#)), and tense interpretation ([Lascarides and Asher 1993](#), [Kehler 1994](#), [Kehler 2000](#)). An extensive investigation into the relationship between coherence relations and discourse connectives can be found in [Knott and Dale \(1994\)](#).

Useful surveys of discourse processing and structure include [Stede \(2011\)](#) and [Webber et al. \(2012\)](#).

Exercises

- 23.1** Finish the Centering Theory processing of the last two utterances of (23.30), and show how (23.29) would be processed. Does the algorithm indeed mark (23.29) as less coherent?
- 23.2** Select an editorial column from your favorite newspaper, and determine the discourse structure for a 10–20 sentence portion. What problems did you encounter? Were you helped by superficial cues the speaker included (e.g., discourse connectives) in any places?

CHAPTER

24

Summarization

Placeholder

CHAPTER

25

Question Answering

The quest for knowledge is deeply human, and so it is not surprising that practically as soon as there were computers we were asking them questions. By the early 1960s, systems used the two major paradigms of question answering—**information-retrieval-based** and **knowledge-based**—to answer questions about baseball statistics or scientific facts. Even imaginary computers got into the act. Deep Thought, the computer that Douglas Adams invented in *The Hitchhiker's Guide to the Galaxy*, managed to answer “the Ultimate Question Of Life, The Universe, and Everything”.¹ In 2011, IBM’s Watson question-answering system won the TV game-show *Jeopardy!* using a hybrid architecture that surpassed humans at answering questions like

WILLIAM WILKINSON’S “AN ACCOUNT OF THE PRINCIPALITIES OF WALLACHIA AND MOLDOVIA” INSPIRED THIS AUTHOR’S MOST FAMOUS NOVEL²

Most question answering systems focus on **factoid questions**, questions that can be answered with simple facts expressed in short texts. The answers to the questions below can be expressed by a personal name, temporal expression, or location:

- (25.1) Who founded Virgin Airlines?
- (25.2) What is the average age of the onset of autism?
- (25.3) Where is Apple Computer based?

In this chapter we describe the two major paradigms for factoid question answering. Information-retrieval or **IR-based question answering** relies on the vast quantities of textual information on the web or in collections like PubMed. Given a user question, information retrieval techniques first find relevant documents and passages. Then systems (feature-based, neural, or both) use **reading comprehension** algorithms to read these retrieved documents or passages and draw an answer directly from **spans of text**.

In the second paradigm, **knowledge-based question answering**, a system instead builds a **semantic representation** of the query, mapping *What states border Texas?* to the logical representation: $\lambda x.state(x) \wedge borders(x, \text{texas})$, or *When was Ada Lovelace born?* to the gapped relation: **birth-year** (Ada Lovelace, ?x). These **meaning representations** are then used to query databases of facts.

Finally, large industrial systems like the DeepQA system in IBM’s Watson are often **hybrids**, using both **text datasets** and **structured knowledge bases** to answer questions. DeepQA finds many candidate answers in both knowledge bases and in textual sources, and then scores each candidate answer using knowledge sources like geospatial databases, taxonomical classification, or other textual sources.

We describe IR-based approaches (including neural reading comprehension systems) in the next section, followed by sections on knowledge-based systems, on Watson Deep QA, and a discussion of evaluation.

¹ The answer was 42, but unfortunately the details of the question were never revealed.

² The answer, of course, is ‘Who is Bram Stoker’, and the novel was *Dracula*.

25.1 IR-based Factoid Question Answering

The goal of information retrieval based question answering is to answer a user's question by finding short text segments on the web or some other collection of documents. Figure 25.1 shows some sample factoid questions and their answers.

Question	Answer
Where is the Louvre Museum located?	in Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	the yuan
What kind of nuts are used in marzipan?	almonds
What instrument does Max Roach play?	drums
What's the official language of Algeria?	Arabic
How many pounds are there in a stone?	14

Figure 25.1 Some sample factoid questions and their answers.

Figure 25.2 shows the three phases of an IR-based factoid question-answering system: question processing, passage retrieval and ranking, and answer extraction.



Figure 25.2 IR-based factoid question answering has three stages: question processing, passage retrieval, and answer processing.

25.1.1 Question Processing

The main goal of the question-processing phase is to extract the **query**: the keywords passed to the IR system to match potential documents. Some systems additionally extract further information such as:

- **answer type**: the entity type (person, location, time, etc.) of the answer.
- **focus**: the string of words in the question that is likely to be replaced by the answer in any answer string found.
- **question type**: is this a definition question, a math question, a list question?

For example, for the question *Which US state capital has the largest population?* the query processing might produce:

query: “US state capital has the largest population”

answer type: city

focus: state capital

In the next two sections we summarize the two most commonly used tasks, query formulation and answer type detection.

25.1.2 Query Formulation

Query formulation is the task of creating a query—a list of tokens—to send to an information retrieval system to retrieve documents that might contain answer strings.

For question answering from the web, we can simply pass the entire question to the web search engine, at most perhaps leaving out the question word (*where*, *when*, etc.). For question answering from smaller sets of documents like corporate information pages or Wikipedia, we still use an IR engine to index and search our documents, generally using standard tf-idf cosine matching, but we might need to do more processing. For example, for searching Wikipedia, it helps to compute tf-idf over bigrams rather than unigrams in the query and document (Chen et al., 2017). Or we might need to do query expansion, since while on the web the answer to a question might appear in many different forms, one of which will probably match the question, in smaller document sets an answer might appear only once. Query expansion methods can add query terms in hopes of matching the particular form of the answer as it appears, like adding morphological variants of the content words in the question, or synonyms from a thesaurus.

query reformulation

A query formulation approach that is sometimes used for questioning the web is to apply **query reformulation** rules to the query. The rules rephrase the question to make it look like a substring of possible declarative answers. The question “*when was the laser invented?*” might be reformulated as “*the laser was invented*”; the question “*where is the Valley of the Kings?*” as “*the Valley of the Kings is located in*”. Here are some sample handwritten reformulation rules from Lin (2007):

(25.4) *wh-word* did A *verb* B → ... A *verb+ed* B

(25.5) Where is A → A is located in

25.1.3 Answer Types

question classification
answer type

Some systems make use of **question classification**, the task of finding the **answer type**, the named-entity categorizing the answer. A question like “*Who founded Virgin Airlines?*” expects an answer of type PERSON. A question like “*What Canadian city has the largest population?*” expects an answer of type CITY. If we know that the answer type for a question is a person, we can avoid examining every sentence in the document collection, instead focusing on sentences mentioning people.

answer type taxonomy

While answer types might just be the named entities like PERSON, LOCATION, and ORGANIZATION described in Chapter 18, we can also use a larger hierarchical set of answer types called an **answer type taxonomy**. Such taxonomies can be built automatically, from resources like WordNet (Harabagiu et al. 2000, Pasca 2003), or they can be designed by hand. Figure 25.4 shows one such hand-built ontology, the Li and Roth (2005) tagset; a subset is also shown in Fig. 25.3. In this hierarchical tagset, each question can be labeled with a coarse-grained tag like HUMAN or a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on. The HUMAN:DESCRIPTION type is often called a BIOGRAPHY question because the answer is required to give a brief biography of the person rather than just a name.

Question classifiers can be built by hand-writing rules like the following rule from (Hovy et al., 2002) for detecting the answer type BIOGRAPHY:

(25.6) who {is | was | are | were} PERSON

Most question classifiers, however, are based on **supervised learning**, trained on databases of questions that have been hand-labeled with an answer type (Li and Roth, 2002). Either feature-based or neural methods can be used. Feature based

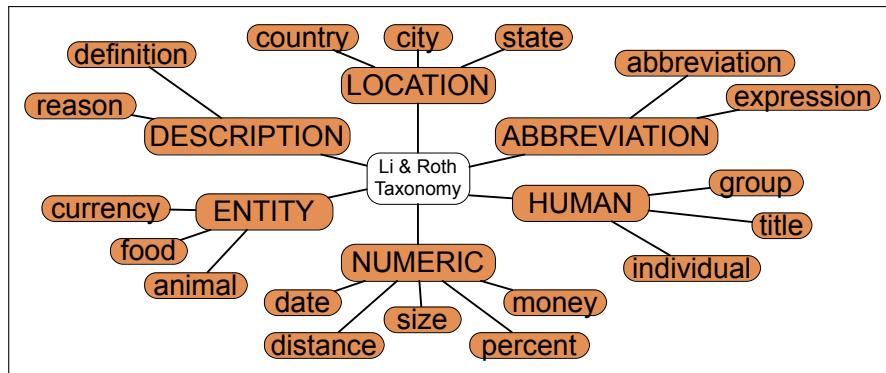


Figure 25.3 A subset of the Li and Roth (2005) answer types.

methods rely on words in the questions and their embeddings, the part-of-speech of each word, and named entities in the questions. Often, a single word in the question gives extra information about the answer type, and its identity is used as a feature. This word is sometimes called the **answer type word** or **question headword**, and may be defined as the headword of the first NP after the question's *wh-word*; headwords are indicated in boldface in the following examples:

- (25.7) Which **city** in China has the largest number of foreign financial companies?
 (25.8) What is the **state flower** of California?

In general, question classification accuracies are relatively high on easy question types like PERSON, LOCATION, and TIME questions; detecting REASON and DESCRIPTION questions can be much harder.

25.1.4 Document and Passage Retrieval

The IR query produced from the question processing stage is sent to an IR engine, resulting in a set of documents ranked by their relevance to the query. Because most answer-extraction methods are designed to apply to smaller regions such as passages, QA systems next divide the top n documents into smaller **passages** such as sections, paragraphs, or sentences. These might be already segmented in the source document or we might need to run a paragraph segmentation algorithm.

The simplest form of **passage retrieval** is then to simply pass along every passage to the answer extraction stage. A more sophisticated variant is to filter the passages by running a named entity or answer type classification on the retrieved passages, discarding passages that don't contain the answer type of the question. It's also possible to use supervised learning to fully rank the remaining passages, using features like:

- The number of **named entities** of the right type in the passage
- The number of **question keywords** in the passage
- The longest exact sequence of question keywords that occurs in the passage
- The rank of the document from which the passage was extracted
- The **proximity** of the keywords from the original query to each other (Pasca 2003, Monz 2004).
- The number of **n -grams** that **overlap** between the passage and the question (Brill et al., 2002).

snippets For question answering from the web we can instead take **snippets** from a Web search engine as the passages.

Tag	Example
ABBREVIATION	
abb	What's the abbreviation for limited partnership?
exp	What does the "c" stand for in the equation E=mc2?
DESCRIPTION	
definition	What are tannins?
description	What are the words to the Canadian National anthem?
manner	How can you get rust stains out of clothing?
reason	What caused the Titanic to sink?
ENTITY	
animal	What are the names of Odin's ravens?
body	What part of your body contains the corpus callosum?
color	What colors make up a rainbow?
creative	In what book can I find the story of Aladdin?
currency	What currency is used in China?
disease/medicine	What does Salk vaccine prevent?
event	What war involved the battle of Chapultepec?
food	What kind of nuts are used in marzipan?
instrument	What instrument does Max Roach play?
lang	What's the official language of Algeria?
letter	What letter appears on the cold-water tap in Spain?
other	What is the name of King Arthur's sword?
plant	What are some fragrant white climbing roses?
product	What is the fastest computer?
religion	What religion has the most members?
sport	What was the name of the ball game played by the Mayans?
substance	What fuel do airplanes use?
symbol	What is the chemical symbol for nitrogen?
technique	What is the best way to remove wallpaper?
term	How do you say " Grandma" in Irish?
vehicle	What was the name of Captain Bligh's ship?
word	What's the singular of dice?
HUMAN	
description	Who was Confucius?
group	What are the major companies that are part of Dow Jones?
ind	Who was the first Russian astronaut to do a spacewalk?
title	What was Queen Victoria's title regarding India?
LOCATION	
city	What's the oldest capital city in the Americas?
country	What country borders the most others?
mountain	What is the highest peak in Africa?
other	What river runs through Liverpool?
state	What states do not have state income tax?
NUMERIC	
code	What is the telephone number for the University of Colorado?
count	About how many soldiers died in World War II?
date	What is the date of Boxing Day?
distance	How long was Mao's 1930s Long March?
money	How much did a McDonald's hamburger cost in 1963?
order	Where does Shanghai rank among world cities in population?
other	What is the population of Mexico?
period	What was the average life expectancy during the Stone Age?
percent	What fraction of a beaver's life is spent swimming?
temp	How hot should the oven be when making Peachy Oat Muffins?
speed	How fast must a spacecraft travel to escape Earth's gravity?
size	What is the size of Argentina?
weight	How many pounds are there in a stone?

Figure 25.4 Question typology from Li and Roth (2002), (2005). Example sentences are from their corpus of 5500 labeled questions. A question can be labeled either with a coarse-grained tag like HUMAN or NUMERIC or with a fine-grained tag like HUMAN:DESCRIPTION, HUMAN:GROUP, HUMAN:IND, and so on.

25.1.5 Answer Extraction

The final stage of question answering is to extract a specific answer from the passage, for example responding *29,029 feet* to a question like “*How tall is Mt. Everest?*”. This task is commonly modeled by **span labeling**: given a passage, identifying the **span** of text which constitutes an answer.

A simple baseline algorithm for answer extraction is to run a named entity tagger on the candidate passage and return whatever span in the passage is the correct answer type. Thus, in the following examples, the underlined named entities would be extracted from the passages as the answer to the HUMAN and DISTANCE-QUANTITY questions:

“Who is the prime minister of India?”

Manmohan Singh, Prime Minister of India, had told left leaders that the deal would not be renegotiated.

“How tall is Mt. Everest?”

The official height of Mount Everest is 29029 feet

Unfortunately, the answers to many questions, such as **DEFINITION** questions, don’t tend to be of a particular named entity type. For this reason modern work on answer extraction uses more sophisticated algorithms, generally based on supervised learning. The next section introduces a simple feature-based classifier, after which we turn to modern neural algorithms.

25.1.6 Feature-based Answer Extraction

Supervised learning approaches to answer extraction train classifiers to decide if a span or a sentence contains an answer. One obviously useful feature is the answer type feature of the above baseline algorithm. Hand-written regular expression patterns also play a role, such as the sample patterns for definition questions in Fig. 25.5.

Pattern	Question	Answer
<AP> such as <QP>	What is autism?	“, developmental disorders such as autism”
<QP>, a <AP>	What is a caldera?	“the Long Valley caldera, a <u>volcanic crater</u> 19 miles long”

Figure 25.5 Some answer-extraction patterns using the **answer phrase** (AP) and **question phrase** (QP) for definition questions (Pasca, 2003).

Other features in such classifiers include:

Answer type match: True if the candidate answer contains a phrase with the correct answer type.

Pattern match: The identity of a pattern that matches the candidate answer.

Number of matched question keywords: How many question keywords are contained in the candidate answer.

Keyword distance: The distance between the candidate answer and query keywords.

Novelty factor: True if at least one word in the candidate answer is novel, that is, not in the query.

Apposition features: True if the candidate answer is an appositive to a phrase containing many question terms. Can be approximated by the number of question terms separated from the candidate answer through at most three words and one comma (Pasca, 2003).

Punctuation location: True if the candidate answer is immediately followed by a comma, period, quotation marks, semicolon, or exclamation mark.

Sequences of question terms: The length of the longest sequence of question terms that occurs in the candidate answer.

25.1.7 N-gram tiling answer extraction

n-gram tiling

An alternative approach to answer extraction, used solely in Web search, is based on **n-gram tiling**, an approach that relies on the **redundancy** of the web (Brill et al. 2002, Lin 2007). This simplified method begins with the snippets returned from the Web search engine, produced by a reformulated query. In the first step, n-gram mining, every unigram, bigram, and trigram occurring in the snippet is extracted and weighted. The weight is a function of the number of snippets in which the n-gram occurred, and the weight of the query reformulation pattern that returned it. In the n-gram filtering step, n-grams are scored by how well they match the predicted answer type. These scores are computed by handwritten filters built for each answer type. Finally, an n-gram tiling algorithm concatenates overlapping n-gram fragments into longer answers. A standard greedy method is to start with the highest-scoring candidate and try to tile each other candidate with this candidate. The best-scoring concatenation is added to the set of candidates, the lower-scoring candidate is removed, and the process continues until a single answer is built.

25.1.8 Neural Answer Extraction

Neural network approaches to answer extraction draw on the intuition that a question and its answer are semantically similar in some appropriate way. As we'll see, this intuition can be fleshed out by computing an embedding for the question and an embedding for each token of the passage, and then selecting passage spans whose embeddings are closest to the question embedding.

Reading Comprehension

reading comprehension

Neural answer extractors are often designed in the context of the **reading comprehension** task. It was Hirschman et al. (1999) who first proposed to take children's reading comprehension tests—pedagogical instruments in which a child is given a passage to read and must answer questions about it—and use them to evaluate machine text comprehension algorithm. They acquired a corpus of 120 passages with 5 questions each designed for 3rd-6th grade children, built an answer extraction system, and measured how well the answers given by their system corresponded to the answer key from the test's publisher.

Since then reading comprehension has become both a task in itself, as a useful way to measure natural language understanding performance, but also as (sometimes called the **reader** component of question answerers).

SQuAD

Reading Comprehension Datasets. Modern reading comprehension systems tend to use collections of questions that are designed specifically for NLP, and so are large enough for training supervised learning systems. For example the **Stanford Question Answering Dataset (SQuAD)** consists of passages from Wikipedia and associated questions whose answers are spans from the passage, as well as some questions that are designed to be unanswerable (Rajpurkar et al. 2016, Rajpurkar et al. 2018); a total of just over 150,000 questions. Fig. 25.6 shows a (shortened) excerpt from a SQuAD 2.0 passage together with three questions and their answer spans.

Beyoncé Giselle Knowles-Carter (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".
Q: "In what city and state did Beyoncé grow up?"
A: "Houston, Texas"
Q: "What areas did Beyoncé compete in when she was growing up?"
A: "singing and dancing"
Q: "When did Beyoncé release Dangerously in Love?"
A: "2003"

Figure 25.6 A (Wikipedia) passage from the SQuAD 2.0 dataset (Rajpurkar et al., 2018) with 3 sample questions and the labeled answer spans.

sentence
selection

SQuAD was built by having humans write questions for a given Wikipedia passage and choose the answer span. Other datasets used similar techniques; the NewsQA dataset consists of 100,000 question-answer pairs from CNN news articles. For other datasets like WikiQA the span is the entire sentence containing the answer (Yang et al., 2015); the task of choosing a sentence rather than a smaller answer span is sometimes called the sentence selection task.

25.1.9 A bi-LSTM-based Reading Comprehension Algorithm

Neural algorithms for reading comprehension are given a question q of l tokens q_1, \dots, q_l and a passage p of m tokens p_1, \dots, p_m . Their goal is to compute, for each token p_i the probability $p_{\text{start}}(i)$ that p_i is the start of the answer span, and the probability $p_{\text{end}}(i)$ that p_i is the end of the answer span.

Fig. 25.7 shows the architecture of the Document Reader component of the DrQA system of Chen et al. (2017). Like most such systems, DrQA builds an embedding for the question, builds an embedding for each token in the passage, computes a similarity function between the question and each passage word in context, and then uses the question-passage similarity scores to decide where the answer span starts and ends.

Let's consider the algorithm in detail, following closely the description in Chen et al. (2017). The question is represented by a single embedding \mathbf{q} , which is a weighted sum of representations for each question word q_i . It is computed by passing the series of embeddings $P\mathbf{E}(q_1), \dots, P\mathbf{E}(q_l)$ of question words through an RNN (such as a bi-LSTM shown in Fig. 25.7). The resulting hidden representations $\{\mathbf{q}_1, \dots, \mathbf{q}_l\}$ are combined by a weighted sum

$$\mathbf{q} = \sum_j b_j \mathbf{q}_j \quad (25.9)$$

The weight b_j is a measure of the relevance of each question word, and relies on a learned weight vector \mathbf{w} :

$$b_j = \frac{\exp(\mathbf{w} \cdot \mathbf{q}_j)}{\sum_{j'} \exp(\mathbf{w} \cdot \mathbf{q}'_{j'})} \quad (25.10)$$

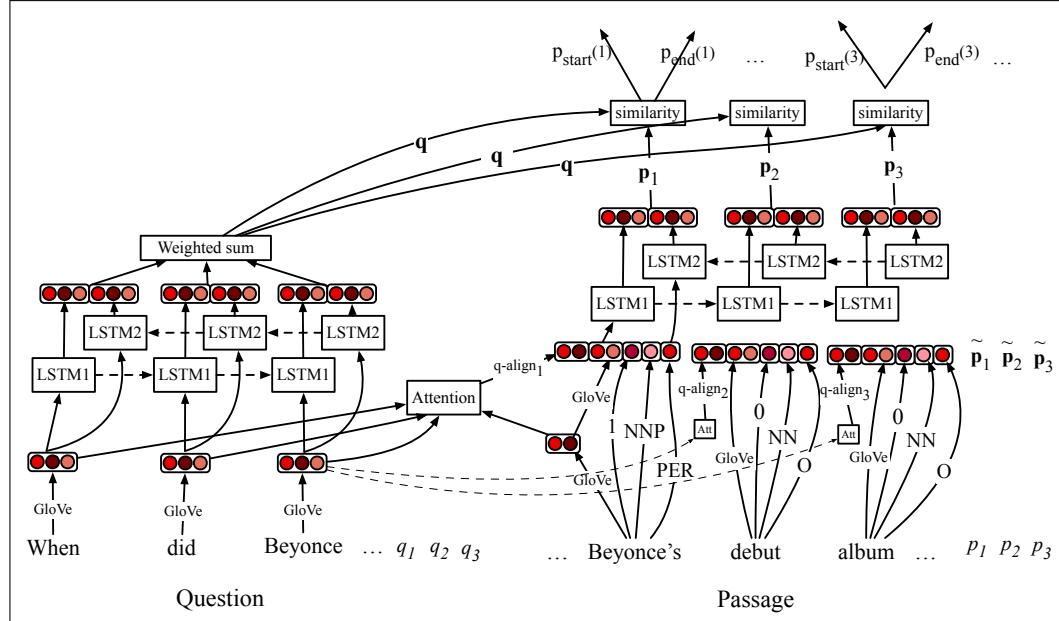


Figure 25.7 The question answering system of Chen et al. (2017), considering part of the question *When did Beyoncé release Dangerously in Love?* and the passage starting *Beyoncé’s debut album, Dangerously in Love (2003)*.

To compute the passage embedding $\{\mathbf{p}_1, \dots, \mathbf{p}_m\}$ we first form an input representation $\tilde{\mathbf{p}} = \{\tilde{\mathbf{p}}_1, \dots, \tilde{\mathbf{p}}_m\}$ by concatenating four components:

- An embedding for each word $\mathbf{E}(p_i)$ such as from GLoVe (Pennington et al., 2014).
- Token features like the part of speech of p_i , or the named entity tag of p_i , from running POS or NER taggers.
- Exact match features representing whether the passage word p_i occurred in the question: $\mathbb{1}(p_i \in q)$. Separate exact match features might be used for lemmatized or lower-cased versions of the tokens.
- Aligned question embedding: In addition to the exact match features, many QA systems use an attention mechanism to give a more sophisticated model of similarity between the passage and question words, such as similar but non-identical words like *release* and *singles*. For example a weighted similarity $\sum_j a_{i,j} \mathbf{E}(q_j)$ can be used, where the attention weight $a_{i,j}$ encodes the similarity between p_i and each question word q_j . This attention weight can be computed as the dot product between functions α of the word embeddings of the question and passage:

$$q_{i,j} = \frac{\exp(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_j)))}{\sum_{j'} \exp(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q'_j)))} \quad (25.11)$$

$\alpha(\cdot)$ can be a simple feed forward network.

We then pass $\tilde{\mathbf{p}}$ through a biLSTM:

$$\{\mathbf{p}_1, \dots, \mathbf{p}_m\} = RNN(\{\tilde{\mathbf{p}}_1, \dots, \tilde{\mathbf{p}}_m\}) \quad (25.12)$$

The result of the previous two steps is a single question embedding \mathbf{q} and a representation for each word in the passage $\{\mathbf{p}_1, \dots, \mathbf{p}_m\}$. In order to find the answer span, we can train two separate classifiers, one to compute for each p_i the

probability $p_{start}(i)$ that p_i is the start of the answer span, and one to compute the probability $p_{end}(i)$. While the classifiers could just take the dot product between the passage and question embeddings as input, it turns out to work better to learn a more sophisticated similarity function, like a bilinear attention layer \mathbf{W} :

$$\begin{aligned} p_{start}(i) &\propto \exp(\mathbf{p}_i \mathbf{W}_s \mathbf{q}) \\ p_{end}(i) &\propto \exp(\mathbf{p}_i \mathbf{W}_e \mathbf{q}) \end{aligned} \quad (25.13)$$

These neural answer extractors can be trained end-to-end by using datasets like SQuAD.

25.1.10 BERT-based Question Answering

The power of contextual embeddings allow question answering models based on BERT contextual embeddings and the transformer architecture to achieve even higher accuracy (Fig. 25.8).



Figure 25.8 The BERT model for span-based question answering from reading comprehension-based question answering tasks. Figure after Devlin et al. (2019).

Recall from Chapter 10 that BERT represents two input strings as a sequence of wordpiece tokens separated with a [SEP] token. The pre-trained BERT model will produce an output token embedding T'_i for every paragraph token i' . For span-based question answering, we represent the question as the first sequence and, the paragraph as the second sequence. We'll also need to add some structure to the output head that will be trained in the fine-tuning phase. We'll add two new embeddings: a span-start embedding S and a span-end embedding E . To get a span-start probability for each output token T'_i , we compute the dot product between S and T'_i and then normalize over all tokens T'_i in the paragraph:

$$Pstart_i = \frac{e^{S \cdot T_i}}{\sum_j e^{S \cdot T_j}} \quad (25.14)$$

We do the analogous thing to compute a span-end probability:

$$Pend_i = \frac{e^{E \cdot T_i}}{\sum_j e^{E \cdot T_j}} \quad (25.15)$$

The score of a candidate span from position i to j is $S \cdot T_i + E \cdot T_j$, and the highest scoring span in which $j \geq i$ is chosen is the model prediction. The training objective for fine-tuning is the sum of the log-likelihoods of the correct start and end positions for each observation.

25.2 Knowledge-based Question Answering

While an enormous amount of information is encoded in the vast amount of text on the web, information obviously also exists in more structured forms. We use the term **knowledge-based question answering** for the idea of answering a natural language question by mapping it to a query over a structured database. Like the text-based paradigm for question answering, this approach dates back to the earliest days of natural language processing, with systems like BASEBALL (Green et al., 1961) that answered questions from a structured database of baseball games and stats.

Systems for mapping from a text string to any logical form are called **semantic parsers**. Semantic parsers for question answering usually map either to some version of predicate calculus or a query language like SQL or SPARQL, as in the examples in Fig. 25.9.

Question	Logical form
When was Ada Lovelace born?	<code>birth-year (Ada Lovelace, ?x)</code>
What states border Texas?	$\lambda x. \text{state}(x) \wedge \text{borders}(x, \text{texas})$
What is the largest state	$\text{argmax}(\lambda x. \text{state}(x), \lambda x. \text{size}(x))$
How many people survived the sinking of the Titanic	$(\text{count} (\text{!fb: event disaster survivors fb:en.sinking_of_the_titanic}))$

Figure 25.9 Sample logical forms produced by a semantic parser for question answering. These range from simple relations like `birth-year`, or relations normalized to databases like Freebase, to full predicate calculus.

The logical form of the question is thus either in the form of a query or can easily be converted into one. The database can be a full relational database, or simpler structured databases like sets of **RDF triples**. Recall from Chapter 18 that an RDF triple is a 3-tuple, a predicate with two arguments, expressing some simple relation or proposition. Popular ontologies like Freebase (Bollacker et al., 2008) or DBpedia (Bizer et al., 2009) have large numbers of triples derived from Wikipedia **infoboxes**, the structured tables associated with certain Wikipedia articles.

The simplest formation of the knowledge-based question answering task is to answer factoid questions that ask about one of the missing arguments in a triple. Consider an RDF triple like the following:

subject	predicate	object
Ada Lovelace	<code>birth-year</code>	1815

This triple can be used to answer text questions like ‘When was Ada Lovelace born?’ or ‘Who was born in 1815?’. Question answering in this paradigm requires mapping from textual strings like “When was ... born” to canonical relations in the knowledge base like `birth-year`. We might sketch this task as:

“When was Ada Lovelace born?” → `birth-year (Ada Lovelace, ?x)`
 “What is the capital of England?” → `capital-city(?x, England)`

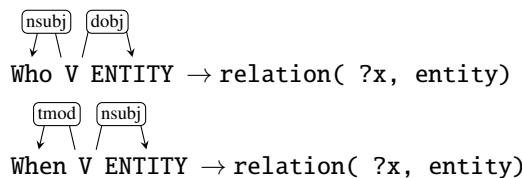
25.2.1 Rule-based Methods

For relations that are very frequent, it may be worthwhile to write handwritten rules to extract relations from the question, just as we saw in Section 18.2. For example, to extract the birth-year relation, we could write patterns that search for the question word *When*, a main verb like *born*, and then extract the named entity argument of the verb.

25.2.2 Supervised Methods

In some cases we have supervised data, consisting of a set of questions paired with their correct logical form like the examples in Fig. 25.9. The task is then to take those pairs of training tuples and produce a system that maps from new questions to their logical forms.

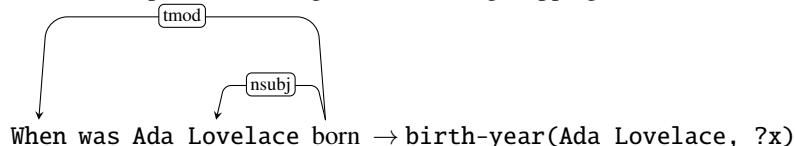
Most supervised algorithms for learning to answer these simple questions about relations first parse the questions and then align the parse trees to the logical form. Generally these systems bootstrap by having a small set of rules for building this mapping, and an initial lexicon as well. For example, a system might have built-in strings for each of the entities in the system (Texas, Ada Lovelace), and then have simple default rules mapping fragments of the question parse tree to particular relations:



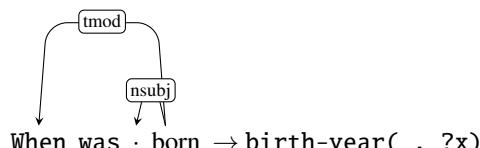
Then given these rules and the lexicon, a training tuple like the following:

“When was Ada Lovelace born?” → *birth-year* (Ada Lovelace, ?x)

would first be parsed, resulting in the following mapping.



From many pairs like this, we could induce mappings between pieces of parse fragment, such as the mapping between the parse fragment on the left and the relation on the right:



A supervised system would thus parse each tuple in the training set and induce a bigger set of such specific rules, allowing it to map unseen examples of “When was X born?” questions to the *birth-year* relation. Rules can furthermore be associated with counts based on the number of times the rule is used to parse the training data. Like rule counts for probabilistic grammars, these can be normalized into probabilities. The probabilities can then be used to choose the highest probability parse for sentences with multiple semantic interpretations.

The supervised approach can be extended to deal with more complex questions that are not just about single relations. Consider the question *What is the biggest state bordering Texas?* —taken from the GeoQuery database of questions on U.S. Geography ([Zelle and Mooney, 1996](#))—with the semantic form: $\operatorname{argmax}(\lambda x.\operatorname{state}(x) \wedge \operatorname{borders}(x,\text{texas}), \lambda x.\operatorname{size}(x))$ This question has much more complex structures than the simple single-relation questions we considered above, such as the argmax function, the mapping of the word *biggest* to *size* and so on. [Zettlemoyer and Collins \(2005\)](#) shows how more complex default rules (along with richer syntactic structures) can be used to learn to map from text sentences to more complex logical forms. The rules take the training set’s pairings of sentence and meaning as above and use the complex rules to break each training example down into smaller tuples that can then be recombined to parse new sentences.

25.2.3 Dealing with Variation: Semi-Supervised Methods

Because it is difficult to create training sets with questions labeled with their meaning representation, supervised datasets can’t cover the wide variety of forms that even simple factoid questions can take. For this reason most techniques for mapping factoid questions to the canonical relations or other structures in knowledge bases find some way to make use of textual redundancy.

The most common source of redundancy, of course, is the web, which contains vast numbers of textual variants expressing any relation. For this reason, most methods make some use of web text, either via semi-supervised methods like **distant supervision** or unsupervised methods like **open information extraction**, both introduced in Chapter 18. For example the REVERB open information extractor ([Fader et al., 2011](#)) extracts billions of (subject, relation, object) triples of strings from the web, such as (“Ada Lovelace”, “was born in”, “1815”). By **aligning** these strings with a canonical knowledge source like Wikipedia, we create new relations that can be queried while simultaneously learning to map between the words in question and canonical relations.

To align a REVERB triple with a canonical knowledge source we first align the arguments and then the predicate. Recall from Chapter 22 that linking a string like “Ada Lovelace” with a Wikipedia page is called **entity linking**; we thus represent the concept ‘Ada Lovelace’ by a unique identifier of a Wikipedia page. If this subject string is not associated with a unique page on Wikipedia, we can disambiguate which page is being sought, for example by using the cosine distance between the triple string (‘Ada Lovelace was born in 1815’) and each candidate Wikipedia page. Date strings like ‘1815’ can be turned into a normalized form using standard tools for temporal normalization like SUTime ([Chang and Manning, 2012](#)). Once we’ve aligned the arguments, we align the predicates. Given the Freebase relation `people.person.birthdate(ada_lovelace, 1815)` and the string ‘Ada Lovelace was born in 1815’, having linked Ada Lovelace and normalized 1815, we learn the mapping between the string ‘was born in’ and the relation `people.person.birthdate`. In the simplest case, this can be done by aligning the relation with the string of words in between the arguments; more complex alignment algorithms like IBM Model 1 (Chapter 11) can be used. Then if a phrase aligns with a predicate across many entities, it can be extracted into a lexicon for mapping questions to relations.

Here are some examples from such a resulting lexicon, produced by [Berant et al. \(2013\)](#), giving many variants of phrases that align with the Freebase relation `country.capital` between a country and its capital city:

Other useful sources of linguistic redundancy are paraphrase databases. For ex-

capital of	capital city of	become capital of
capitol of	national capital of	official capital of
political capital of	administrative capital of	beautiful capital of
capitol city of	remain capital of	make capital of
political center of	bustling capital of	capital city in
cosmopolitan capital of	move its capital to	modern capital of
federal capital of	beautiful capital city of	administrative capital city of

Figure 25.10 Some phrases that align with the Freebase relation `country.capital` from Berant et al. (2013).

ample the site wikianswers.com contains millions of pairs of questions that users have tagged as having the same meaning, 18 million of which have been collected in the PARALEX corpus (Fader et al., 2013). Here's an example:

Q: What are the green blobs in plant cells?

Lemmatized synonyms from PARALEX:

what be the green blob in plant cell?
 what be green part in plant cell?
 what be the green part of a plant cell?
 what be the green substance in plant cell?
 what be the part of plant cell that give it green color?
 what cell part do plant have that enable the plant to be give a green color?
 what part of the plant cell turn it green?
 part of the plant cell where the cell get it green color?
 the green part in a plant be call?
 the part of the plant cell that make the plant green be call?

The resulting millions of pairs of question paraphrases can be aligned to each other using MT alignment approaches to create an MT-style phrase table for translating from question phrases to synonymous phrases. These can be used by question answering algorithms to generate all paraphrases of a question as part of the process of finding an answer (Fader et al. 2013, Berant and Liang 2014).

25.3 Using multiple information sources: IBM's Watson

Of course there is no reason to limit ourselves to just text-based or knowledge-based resources for question answering. The Watson system from IBM that won the Jeopardy! challenge in 2011 is an example of a system that relies on a wide variety of resources to answer questions.

Figure 25.11 shows the 4 stages of the DeepQA system that is the question answering component of Watson.

The first stage is **question processing**. The DeepQA system runs parsing, named entity tagging, and relation extraction on the question. Then, like the text-based systems in Section 25.1, the DeepQA system extracts the **focus**, the **answer type** (also called the **lexical answer type** or **LAT**), and performs **question classification** and **question sectioning**.

Consider these Jeopardy! examples, with a category followed by a question:

Poets and Poetry: **He** was a bank clerk in the Yukon before he published “Songs of a Sourdough” in 1907.

THEATRE: A new play based on **this Sir Arthur Conan Doyle canine**

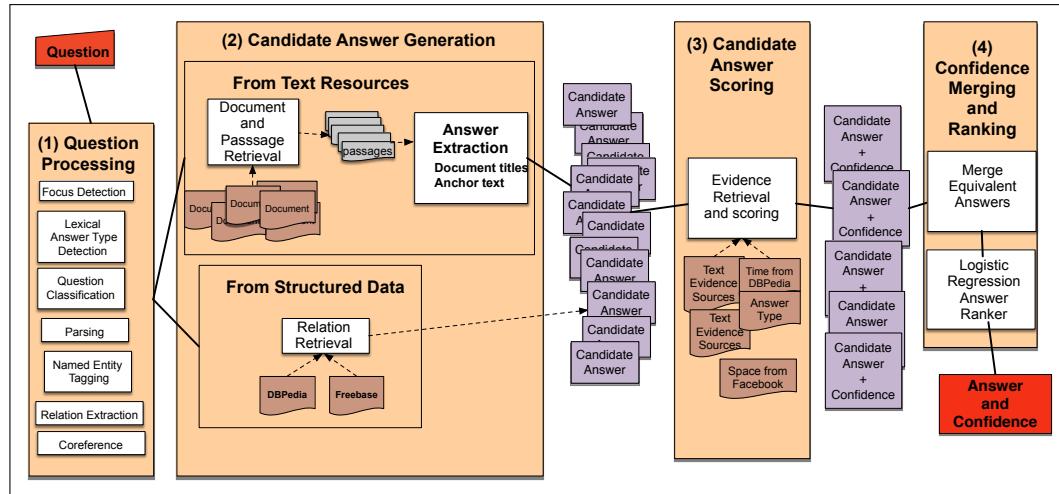


Figure 25.11 The 4 broad stages of Watson QA: (1) Question Processing, (2) Candidate Answer Generation, (3) Candidate Answer Scoring, and (4) Answer Merging and Confidence Scoring.

classic opened on the London stage in 2007.

The questions are parsed, named entities are extracted (*Sir Arthur Conan Doyle* identified as a PERSON, Yukon as a GEOPOLITICAL ENTITY, “Songs of a Sourdough” as a COMPOSITION), coreference is run (*he* is linked with *clerk*) and relations like the following are extracted:

```
authorof(focus,“Songs of a sourdough”)
publish (e1, he, “Songs of a sourdough”)
in (e2, e1, 1907)
temporallink(publish(...), 1907)
```

focus

Next DeepQA extracts the question **focus**, shown in bold in both examples. The focus is the part of the question that co-refers with the answer, used for example to align with a supporting passage. The focus is extracted by handwritten rules—made possible by the relatively stylized syntax of Jeopardy! questions—such as a rule extracting any noun phrase with determiner “this” as in the Conan Doyle example, and rules extracting pronouns like *she*, *he*, *hers*, *him*, as in the poet example.

lexical answer type

The **lexical answer type** (shown in blue above) is a word or words which tell us something about the semantic type of the answer. Because of the wide variety of questions in Jeopardy!, Jeopardy! uses a far larger set of answer types than the sets for standard factoid algorithms like the one shown in Fig. 25.4. Even a large set of named entity tags is insufficient to define a set of answer types. The DeepQA team investigated a set of 20,000 questions and found that a named entity tagger with over 100 named entity types covered less than half the types in these questions. Thus DeepQA extracts a wide variety of words to be answer types; roughly 5,000 lexical answer types occurred in the 20,000 questions they investigated, often with multiple answer types in each question.

These lexical answer types are again extracted by rules: the default rule is to choose the syntactic headword of the focus. Other rules improve this default choice. For example additional lexical answer types can be words in the question that are coreferent with or have a particular syntactic relation with the focus, such as headwords of appositives or predicative nominatives of the focus. In some cases even the Jeopardy! category can act as a lexical answer type, if it refers to a type of entity

that is compatible with the other lexical answer types. Thus in the first case above, *he*, *poet*, and *clerk* are all lexical answer types. In addition to using the rules directly as a classifier, they can instead be used as features in a logistic regression classifier that can return a probability as well as a lexical answer type.

Note that answer types function quite differently in DeepQA than the purely IR-based factoid question answerers. In the algorithm described in Section 25.1, we determine the answer type, and then use a strict filtering algorithm only considering text strings that have exactly that type. In DeepQA, by contrast, we extract lots of answers, unconstrained by answer type, and a set of answer types, and then in the later ‘candidate answer scoring’ phase, we simply score how well each answer fits the answer types as one of many sources of evidence.

Finally the question is classified by type (definition question, multiple-choice, puzzle, fill-in-the-blank). This is generally done by writing pattern-matching regular expressions over words or parse trees.

In the second **candidate answer generation** stage, we combine the processed question with external documents and other knowledge sources to suggest many candidate answers. These candidate answers can either be extracted from text documents or from structured knowledge bases.

For structured resources like DBpedia, IMDB, or the triples produced by Open Information Extraction, we can just query these stores with the relation and the known entity, just as we saw in Section 25.2. Thus if we have extracted the relation `authorof(focus, "Songs of a sourdough")`, we can query a triple store with `authorof(?x, "Songs of a sourdough")` to return the correct author.

The method for extracting answers from text depends on the type of text documents. To extract answers from normal text documents we can do passage search just as we did in Section 25.1. As we did in that section, we need to generate a query from the question; for DeepQA this is generally done by eliminating stop words, and then upweighting any terms which occur in any relation with the focus. For example from this query:

MOVIE-“ING”: Robert Redford and Paul Newman starred in this depression-era grifter flick. (*Answer: “The Sting”*)

the following weighted query might be extracted:

(2.0 Robert Redford) (2.0 Paul Newman) star depression era grifter (1.5 flick)

The query can now be passed to a standard IR system. DeepQA also makes use of the convenient fact that the vast majority of Jeopardy! answers are the title of a Wikipedia document. To find these titles, we can do a second text retrieval pass specifically on Wikipedia documents. Then instead of extracting passages from the retrieved Wikipedia document, we directly return the titles of the highly ranked retrieved documents as the possible answers.

anchor texts

Once we have a set of passages, we need to extract candidate answers. If the document happens to be a Wikipedia page, we can just take the title, but for other texts, like news documents, we need other approaches. Two common approaches are to extract all **anchor texts** in the document (anchor text is the text between `<a>` and `` used to point to a URL in an HTML page), or to extract all noun phrases in the passage that are Wikipedia document titles.

The third **candidate answer scoring** stage uses many sources of evidence to score the candidates. One of the most important is the lexical answer type. DeepQA includes a system that takes a candidate answer and a lexical answer type and returns a score indicating whether the candidate answer can be interpreted as a subclass or

instance of the answer type. Consider the candidate “difficulty swallowing” and the lexical answer type “manifestation”. DeepQA first matches each of these words with possible entities in ontologies like DBpedia and WordNet. Thus the candidate “difficulty swallowing” is matched with the DBpedia entity “Dysphagia”, and then that instance is mapped to the WordNet type “Symptom”. The answer type “manifestation” is mapped to the WordNet type “Condition”. The system looks for a link of hyponymy, instance-of or synonymy between these two types; in this case a hyponymy relation is found between “Symptom” and “Condition”.

Other scorers are based on using time and space relations extracted from DBpedia or other structured databases. For example, we can extract temporal properties of the entity (when was a person born, when died) and then compare to time expressions in the question. If a time expression in the question occurs chronologically before a person was born, that would be evidence against this person being the answer to the question.

Finally, we can use text retrieval to help retrieve evidence supporting a candidate answer. We can retrieve passages with terms matching the question, then replace the focus in the question with the candidate answer and measure the overlapping words or ordering of the passage with the modified question.

The output of this stage is a set of candidate answers, each with a vector of scoring features.

The final **answer merging and scoring** step first merges candidate answers that are equivalent. Thus if we had extracted two candidate answers *JFK*. and *John F. Kennedy*, this stage would merge the two into a single candidate. Synonym dictionaries are a useful resource that are created by listing all anchor text strings that point to the same Wikipedia page; such dictionaries give large numbers of synonyms for each Wikipedia title — e.g., *JFK*, *John F. Kennedy*, *John Fitzgerald Kennedy*, *Senator John F. Kennedy*, *President Kennedy*, *Jack Kennedy*, etc. ([Spitkovsky and Chang, 2012](#)). For common nouns, we can use morphological parsing to merge candidates which are morphological variants.

We then merge the evidence for each variant, combining the scoring feature vectors for the merged candidates into a single vector.

Now we have a set of candidates, each with a feature vector. A classifier takes each feature vector and assigns a confidence value to this candidate answer. The classifier is trained on thousands of candidate answers, each labeled for whether it is correct or incorrect, together with their feature vectors, and learns to predict a probability of being a correct answer. Since, in training, there are far more incorrect answers than correct answers, we need to use one of the standard techniques for dealing with very imbalanced data. DeepQA uses *instance weighting*, assigning an instance weight of .5 for each incorrect answer example in training. The candidate answers are then sorted by this confidence value, resulting in a single best answer.³

In summary, we’ve seen in the four stages of DeepQA that it draws on the intuitions of both the IR-based and knowledge-based paradigms. Indeed, Watson’s architectural innovation is its reliance on proposing a very large number of candidate answers from both text-based and knowledge-based sources and then developing a wide variety of evidence features for scoring these candidates—again both text-based and knowledge-based. See the papers mentioned at the end of the chapter for more details.

³ The merging and ranking is actually run iteratively; first the candidates are ranked by the classifier, giving a rough first value for each candidate answer, then that value is used to decide which of the variants of a name to select as the merged answer, then the merged answers are re-ranked.

25.4 Evaluation of Factoid Answers

mean reciprocal rank
MRR

A common evaluation metric for factoid question answering, introduced in the TREC Q/A track in 1999, is **mean reciprocal rank**, or **MRR**. MRR assumes a test set of questions that have been human-labeled with correct answers. MRR also assumes that systems are returning a short **ranked** list of answers or passages containing answers. Each question is then scored according to the reciprocal of the **rank** of the first correct answer. For example if the system returned five answers but the first three are wrong and hence the highest-ranked correct answer is ranked fourth, the reciprocal rank score for that question would be $\frac{1}{4}$. Questions with return sets that do not contain any correct answers are assigned a zero. The score of a system is then the average of the score for each question in the set. More formally, for an evaluation of a system returning a set of ranked answers for a test set consisting of N questions, the MRR is defined as

$$\text{MRR} = \frac{1}{N} \sum_{i=1 \text{ s.t. } \text{rank}_i \neq 0}^N \frac{1}{\text{rank}_i} \quad (25.16)$$

Reading comprehension systems on datasets like SQuAD are often evaluated using two metrics, both ignoring punctuation and articles (*a, an, the*) (Rajpurkar et al., 2016):

- Exact match: The percentage of predicted answers that match the gold answer exactly.
- F₁ score: The average overlap between predicted and gold answers. Treat the prediction and gold as a bag of tokens, and compute F₁, averaging the F₁ over all questions.

A number of test sets are available for question answering. Early systems used the TREC QA dataset; questions and handwritten answers for TREC competitions from 1999 to 2004 are publicly available. TriviaQA (Joshi et al., 2017) has 650K question-answer evidence triples, from 95K hand-created question-answer pairs together with on average six supporting evidence documents collected retrospectively from Wikipedia and the Web.

Another family of datasets starts from WEBQUESTIONS (Berant et al., 2013), which contains 5,810 questions asked by web users, each beginning with a wh-word and containing exactly one entity. Questions are paired with handwritten answers drawn from the Freebase page of the question’s entity. WEBQUESTONSSP (Yih et al., 2016) augments WEBQUESTIONS with human-created semantic parses (SPARQL queries) for those questions answerable using Freebase. COMPLEXWEBQUESTIONS augments the dataset with compositional and other kinds of complex questions, resulting in 34,689 questions, along with answers, web snippets, and SPARQL queries. (Talmor and Berant, 2018).

There are a wide variety of datasets for training and testing reading comprehension/answer extraction in addition to the SQuAD (Rajpurkar et al., 2016) and WikiQA (Yang et al., 2015) datasets discussed on page 473. The NarrativeQA (Kočiský et al., 2018) dataset, for example, has questions based on entire long documents like books or movie scripts, while the Question Answering in Context (QuAC) dataset (Choi et al., 2018) has 100K questions created by two crowd workers who are asking and answering questions about a hidden Wikipedia text.

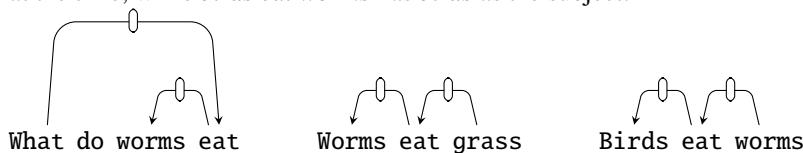
Others take their structure from the fact that reading comprehension tasks designed for children tend to be multiple choice, with the task being to choose among the given answers. The MCTest dataset uses this structure, with 500 fictional short stories created by crowd workers with questions and multiple choice answers (Richardson et al., 2013). The AI2 Reasoning Challenge (ARC) (Clark et al., 2018), has questions that are designed to be hard to answer from simple lexical methods:

- Which property of a mineral can be determined just by looking at it?
 (A) luster [correct] (B) mass (C) weight (D) hardness

This ARC example is difficult because the correct answer *luster* is unlikely to cooccur frequently on the web with phrases like *looking at it*, while the word *mineral* is highly associated with the incorrect answer *hardness*.

Bibliographical and Historical Notes

Question answering was one of the earliest NLP tasks, and early versions of the text-based and knowledge-based paradigms were developed by the very early 1960s. The text-based algorithms generally relied on simple parsing of the question and of the sentences in the document, and then looking for matches. This approach was used very early on (Phillips, 1960) but perhaps the most complete early system, and one that strikingly prefigures modern relation-based systems, was the Protosynthex system of Simmons et al. (1964). Given a question, Protosynthex first formed a query from the content words in the question, and then retrieved candidate answer sentences in the document, ranked by their frequency-weighted term overlap with the question. The query and each retrieved sentence were then parsed with dependency parsers, and the sentence whose structure best matches the question structure selected. Thus the question *What do worms eat?* would match *worms eat grass*: both have the subject *worms* as a dependent of *eat*, in the version of dependency grammar used at the time, while *birds eat worms* has *birds* as the subject:



The alternative knowledge-based paradigm was implemented in the BASEBALL system (Green et al., 1961). This system answered questions about baseball games like “Where did the Red Sox play on July 7” by querying a structured database of game information. The database was stored as a kind of attribute-value matrix with values for attributes of each game:

```

Month = July
Place = Boston
Day = 7
Game Serial No. = 96
(Team = Red Sox, Score = 5)
(Team = Yankees, Score = 3)
  
```

Each question was constituency-parsed using the algorithm of Zellig Harris's TDAP project at the University of Pennsylvania, essentially a cascade of finite-

state transducers (see the historical discussion in [Joshi and Hopely 1999](#) and [Karttunen 1999](#)). Then in a content analysis phase each word or phrase was associated with a program that computed parts of its meaning. Thus the phrase ‘Where’ had code to assign the semantics Place = ?, with the result that the question “Where did the Red Sox play on July 7” was assigned the meaning

```
Place = ?
Team = Red Sox
Month = July
Day = 7
```

The question is then matched against the database to return the answer. [Simmons \(1965\)](#) summarizes other early QA systems.

LUNAR Another important progenitor of the knowledge-based paradigm for question-answering is work that used predicate calculus as the meaning representation language. The LUNAR system ([Woods et al. 1972](#), [Woods 1978](#)) was designed to be a natural language interface to a database of chemical facts about lunar geology. It could answer questions like *Do any samples have greater than 13 percent aluminum* by parsing them into a logical form

```
(TEST (FOR SOME X16 / (SEQ SAMPLES) : T ; (CONTAIN' X16
(NPR* X17 / (QUOTE AL203)) (GREATERTHAN 13 PCT))))
```

The rise of the web brought the information-retrieval paradigm for question answering to the forefront with the TREC QA track beginning in 1999, leading to a wide variety of factoid and non-factoid systems competing in annual evaluations.

At the same time, [Hirschman et al. \(1999\)](#) introduced the idea of using children’s reading comprehension tests to evaluate machine text comprehension algorithms. They acquired a corpus of 120 passages with 5 questions each designed for 3rd-6th grade children, built an answer extraction system, and measured how well the answers given by their system corresponded to the answer key from the test’s publisher. Their algorithm focused on word overlap as a feature; later algorithms added named entity features and more complex similarity between the question and the answer span ([Riloff and Thelen 2000](#), [Ng et al. 2000](#)).

Neural reading comprehension systems drew on the insight of these early systems that answer finding should focus on question-passage similarity. Many of the architectural outlines of modern systems were laid out in the AttentiveReader ([Hermann et al., 2015](#)). The idea of using passage-aligned question embeddings in the passage computation was introduced by [Lee et al. \(2017\)](#). [Seo et al. \(2017\)](#) achieves high performance by introducing bi-directional attention flow. [Chen et al. \(2017\)](#) and [Clark and Gardner \(2018\)](#) show how to extract answers from entire documents.

The DeepQA component of the Watson system that won the Jeopardy! challenge is described in a series of papers in volume 56 of the IBM Journal of Research and Development; see for example [Ferrucci \(2012\)](#), [Lally et al. \(2012\)](#), [Chu-Carroll et al. \(2012\)](#), [Murdock et al. \(2012b\)](#), [Murdock et al. \(2012a\)](#), [Kalyanpur et al. \(2012\)](#), and [Gondek et al. \(2012\)](#).

Other question-answering tasks include Quiz Bowl, which has timing considerations since the question can be interrupted ([Boyd-Graber et al., 2018](#)). Question answering is also an important function of modern personal assistant dialog systems; see Chapter 26 for more.

Exercises

Les lois de la conversation sont en général de ne s'y appesantir sur aucun objet, mais de passer légèrement, sans effort et sans affectation, d'un sujet à un autre ; de savoir y parler de choses frivoles comme de choses sérieuses

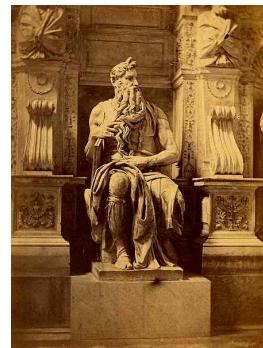
The rules of conversation are, in general, not to dwell on any one subject, but to pass lightly from one to another without effort and without affectation; to know how to speak about trivial topics as well as serious ones;

The 18th C. *Encyclopedia* of Diderot, start of the entry on conversation

conversation
dialogue

conversational agent
dialogue system

The literature of the fantastic abounds in inanimate objects magically endowed with sentience and the gift of speech. From Ovid's statue of Pygmalion to Mary Shelley's Frankenstein, there is something deeply moving about creating something and then having a chat with it. Legend has it that after finishing his sculpture *Moses*, Michelangelo thought it so lifelike that he tapped it on the knee and commanded it to speak. Perhaps this shouldn't be surprising. Language is the mark of humanity and sentience, and **conversation** or **dialogue** is the most fundamental and specially privileged arena of language. It is the first kind of language we learn as children, and for most of us, it is the kind of language we most commonly indulge in, whether we are ordering curry for lunch or buying spinach, participating in business meetings or talking with our families, booking airplane flights or complaining about the weather.



This chapter introduces the fundamental algorithms of **conversational agents**, or **dialogue systems**. These programs communicate with users in natural language (text, speech, or both), and fall into two classes. **Task-oriented dialogue agents** use conversation with users to help complete tasks. Dialogue agents in digital assistants (Siri, Alexa, Google Now/Home, Cortana, etc.), give directions, control appliances, find restaurants, or make calls. Conversational agents can answer questions on corporate websites, interface with robots, and even be used for social good: DoNotPay is a “robot lawyer” that helps people challenge incorrect parking fines, apply for emergency housing, or claim asylum if they are refugees. By contrast, **chatbots** are systems designed for extended conversations, set up to mimic the unstructured conversations or ‘chats’ characteristic of human-human interaction, mainly for entertainment, but also for practical purposes like making task-oriented agents more natural. In Section 26.2 we’ll discuss the three major chatbot architectures : rule-based systems, information retrieval systems, and encoder-decoder models. In Section 26.3 we turn to task-oriented agents, introducing the frame-based architecture (the **GUS** architecture) that underlies most modern task-based systems.

26.1 Properties of Human Conversation

Conversation between humans is an intricate and complex joint activity. Before we attempt to design a conversational agent to converse with humans, it is crucial to understand something about how humans converse with each other. Consider some of the phenomena that occur in the conversation between a human travel agent and a human client excerpted in Fig. 26.1.

C ₁ :	...I need to travel in May.
A ₁ :	And, what day in May did you want to travel?
C ₂ :	OK uh I need to be there for a meeting that's from the 12th to the 15th.
A ₂ :	And you're flying into what city?
C ₃ :	Seattle.
A ₃ :	And what time would you like to leave Pittsburgh?
C ₄ :	Uh hmm I don't think there's many options for non-stop.
A ₄ :	Right. There's three non-stops today.
C ₅ :	What are they?
A ₅ :	The first one departs PGH at 10:00am arrives Seattle at 12:05 their time. The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.
C ₆ :	OK I'll take the 5ish flight on the night before on the 11th.
A ₆ :	On the 11th? OK. Departing at 5:55pm arrives Seattle at 8pm, U.S. Air flight 115.
C ₇ :	OK.
A ₇ :	And you said returning on May 15th?
C ₈ :	Uh, yeah, at the end of the day.
A ₈ :	OK. There's #two non-stops ...#
C ₉ :	#Act...actually #, what day of the week is the 15th?
A ₉ :	It's a Friday.
C ₁₀ :	Uh hmm. I would consider staying there an extra day til Sunday.
A ₁₀ :	OK...OK. On Sunday I have ...

Figure 26.1 Part of a phone conversation between a human travel agent (A) and human client (C). The passages framed by # in A₈ and C₉ indicate overlaps in speech.

TURNS

turn

A dialogue is a sequence of **turns** (A₁, B₁, A₂, and so on), each a single contribution to the dialogue (as if in a game: I take a turn, then you take a turn, then me, and so on). A turn can consist of a sentence (like C₁), although it might be as short as a single word (C₇) or as long as multiple sentences (A₅).

Turn structure has important implications for spoken dialogue. A system has to know when to stop talking; the client interrupts (in A₈ and C₉), so the system must know to stop talking (and that the user might be making a correction). A system also has to know when to start talking. For example, most of the time in conversation, speakers start their turns almost immediately after the other speaker finishes, without a long pause, because people are able to (most of the time) detect when the other person is about to finish talking. Spoken dialogue systems must also detect whether a user is done speaking, so they can process the utterance and respond. This task—called **endpointing** or **endpoint detection**—can be quite challenging because of noise and because people often pause in the middle of turns.

endpointing

Speech Acts

speech acts A key insight into conversation—due originally to the philosopher [Wittgenstein \(1953\)](#) but worked out more fully by [Austin \(1962\)](#)—is that each utterance in a dialogue is a kind of **action** being performed by the speaker. These actions are commonly called **speech acts** or **dialog acts**: here's one taxonomy consisting of 4 major classes ([Bach and Harnish, 1979](#)):

Constatives:	committing the speaker to something's being the case (<i>answering, claiming, confirming, denying, disagreeing, stating</i>)
Directives:	attempts by the speaker to get the addressee to do something (<i>advising, asking, forbidding, inviting, ordering, requesting</i>)
Commissives:	committing the speaker to some future course of action (<i>promising, planning, vowed, betting, opposing</i>)
Acknowledgments:	express the speaker's attitude regarding the hearer with respect to some social action (<i>apologizing, greeting, thanking, accepting an acknowledgment</i>)

A user asking a person or a dialogue system to do something ('Turn up the music') is issuing a DIRECTIVE. Asking a question that requires an answer is also a way of issuing a DIRECTIVE: in a sense when the system says (C₂) "what day in May did you want to travel?" it's as if the system is (very politely) commanding the system to answer. By contrast, a user stating a constraint (like C₁ 'I need to travel in May') is issuing a CONSTATIVE. A user thanking the system is issuing an ACKNOWLEDGMENT. The speech act expresses an important component of the intention of the speaker (or writer) in saying what they said.

Grounding

**common ground
grounding**

A dialogue is not just a series of independent speech acts, but rather a collective act performed by the speaker and the hearer. Like all collective acts, it's important for the participants to establish what they both agree on, called the **common ground** ([Stalnaker, 1978](#)). Speakers do this by **grounding** each other's utterances. Grounding means acknowledging that the hearer has understood the speaker; like an ACK used to confirm receipt in data communications ([Clark, 1996](#)). (People need grounding for non-linguistic actions as well; the reason an elevator button lights up when it's pressed is to acknowledge that the elevator has indeed been called ([Norman, 1988](#).)

Humans constantly ground each other's utterances. We can ground by explicitly saying "OK", as the agent does in A₈ or A₁₀. Or we can ground by repeating what the other person says; in utterance A₁ the agent repeats "in May", demonstrating her understanding to the client. Or notice that when the client answers a question, the agent begins the next question with "And". The "And" implies that the new question is 'in addition' to the old question, again indicating to the client that the agent has successfully understood the answer to the last question.

Subdialogues and Dialogue Structure

conversational analysis

adjacency pair

Conversations have structure. Consider, for example, the local structure between speech acts discussed in the field of **conversational analysis** ([Sacks et al., 1974](#)). QUESTIONS set up an expectation for an ANSWER. PROPOSALS are followed by ACCEPTANCE (or REJECTION). COMPLIMENTS ("Nice jacket!") often give rise to DOWNPLAYERS ("Oh, this old thing?"). These pairs, called **adjacency pairs** are composed of a **first pair part** and a **second pair part** ([Schegloff, 1968](#)), and these expectations can help systems decide what actions to take.

side sequence However, dialogue acts aren't always followed immediately by their second pair part. The two parts can be separated by a **side sequence** (Jefferson 1972) or **subdialogue**. For example utterances C₉ to A₁₀ constitute a **correction subdialogue** (Litman 1985, Litman and Allen 1987, Chu-Carroll and Carberry 1998):

- C₉: #Act... actually#, what day of the week is the 15th?
- A₉: It's a Friday.
- C₁₀: Uh hmm. I would consider staying there an extra day til Sunday.
- A₁₀: OK...OK. On Sunday I have ...

The question in C₉ interrupts the prior discourse, in which the agent was looking for a May 15 return flight. The agent must answer the question and also realize that "I would consider staying...til Sunday" means that the client would probably like to change their plan, and now go back to finding return flights, but for the 17th.

Another side sequence is the **clarification question**, which can form a subdialogue between a REQUEST and a RESPONSE. This is especially common in dialogue systems where speech recognition errors causes the system to have to ask for clarifications or repetitions like the following:

- User: What do you have going to UNKNOWN_WORD on the 5th?
- System: Let's see, going where on the 5th?
- User: Going to Hong Kong.
- System: OK, here are some flights...

presequence

In addition to side-sequences, questions often have **presequences**, like the following example where a user starts with a question about the system's capabilities ("Can you make train reservations") before making a request.

- User: Can you make train reservations?
- System: Yes I can.
- User: Great, I'd like to reserve a seat on the 4pm train to New York.

initiative

Sometimes a conversation is completely controlled by one participant. For example a reporter interviewing a chef might ask questions, and the chef responds. We say that the reporter in this case has the conversational **initiative** (Walker and Whittaker, 1990). In normal human-human dialogue, however, it's more common for initiative to shift back and forth between the participants, as they sometimes answer questions, sometimes ask them, sometimes take the conversations in new directions, sometimes not. You may ask me a question, and then I respond asking you to clarify something you said, which leads the conversation in all sorts of ways. We call such interactions **mixed initiative**.

Mixed initiative, while the norm for human-human conversations, is very difficult for dialogue systems to achieve. It's much easier to design dialogue systems to be passive responders. In the question answering systems we saw in Chapter 25, or in simple search engines, the initiative lies completely with the user. In such **user-initiative** systems, the user specifies a query, and the systems responds. Then the user can specify another query. Alternatively, you may have had the experience of being stuck in a bad dialogue system that asks a question and gives you no opportunity to do anything until you answer it. Such **system-initiative** architectures can be very frustrating.

Inference and Implicature

Inference is also important in dialogue understanding. Consider the client's response C₂, repeated here:

A₁: And, what day in May did you want to travel?

C₂: OK uh I need to be there for a meeting that's from the 12th to the 15th.

Notice that the client does not in fact answer the agent's question. The client merely mentions a meeting at a certain time. What is it that licenses the agent to infer that the client is mentioning this meeting so as to inform the agent of the travel dates?

implicature

relevance

The speaker seems to expect the hearer to draw certain inferences; in other words, the speaker is communicating more information than seems to be present in the uttered words. This kind of example was pointed out by [Grice \(1975, 1978\)](#) as part of his theory of **conversational implicature**. **Implicature** means a particular class of licensed inferences. Grice proposed that what enables hearers to draw these inferences is that conversation is guided by a set of **maxims**, general heuristics that play a guiding role in the interpretation of conversational utterances. One such maxim is the maxim of **relevance** which says that speakers attempt to be relevant, they don't just utter random speech acts. When the client mentions a meeting on the 12th, the agent reasons 'There must be some relevance for mentioning this meeting. What could it be?'. The agent knows that one precondition for having a meeting (at least before Web conferencing) is being at the place where the meeting is held, and therefore that maybe the meeting is a reason for the travel, and if so, then since people like to arrive the day before a meeting, the agent should infer that the flight should be on the 11th.

These subtle characteristics of human conversations (**turns, speech acts, grounding, dialogue structure, initiative, and implicature**) are among the reasons it is difficult to build dialogue systems that can carry on natural conversations with humans. Many of these challenges are active areas of dialogue systems research.

26.2 Chatbots

chatbot

The simplest kinds of dialogue systems are **chatbots**, systems that can carry on extended conversations with the goal of mimicking the unstructured conversations or 'chats' characteristic of informal human-human interaction.

These systems often have an entertainment value, such as Cleverbot, a simple IR-based chatbot capable of carrying on the kinds of conversations shown in Fig. 26.2, or Microsoft's XiaoIce system¹ ([Zhou et al., 2018](#)), a more sophisticated commercial system that chats with people on text messaging platforms (Fig. 26.3). Yet starting from the very first system, ELIZA ([Weizenbaum, 1966](#)), chatbots have also been used for practical purposes, such as testing theories of psychological counseling.

Like practically everything else in language processing, chatbot architectures fall into two classes: rule-based systems and corpus-based systems. Rule-based systems include the early influential ELIZA and PARRY systems. Corpus-based systems mine large datasets of human-human conversations, which can be done by using information retrieval (IR-based systems simply copy a human's response from

¹ XiaoIce is a somewhat confusing translation of the system's Chinese name 小冰, 'Xiao Bing', which means Little Bing or Little Ice.

User:	Do you like Doctor Who?
Cleverbot:	Yes, so funny.
User:	What's your favorite show?
Cleverbot:	I don't have one.
User:	What's your name?
Cleverbot:	Jason I already told you.
User:	No your real name.
Cleverbot:	Stephanie IS my real name.
User:	I thought it was Jason?
Cleverbot:	How did you come to that conclusion.
User:	Because you said so.
Cleverbot:	I have to do my homework.

Figure 26.2 Chatbots: A conversation between a user and Cleverbot, an IR-based chatbot (Carpenter, 2017).

[人类]	小冰你好	[Human]	Hello, XiaoIce.
[小冰]	你好呀，你头像好有趣，感觉 在哪见过	[XiaoIce]	Hello, your profile pic is interesting. Like I've seen it somewhere.
[人类]	是吗	[Human]	Really?
[小冰]	提不是哪部动画里的呀	[XiaoIce]	Is it a comic character?
[人类]	你怎么知道的	[Human]	How do you know that?
[小冰]	你猜	[XiaoIce]	Guess.
[人类]	难道你是真人吗？	[Human]	Are you human?
[小冰]	这个，你和我多聊聊就知道了	[XiaoIce]	Well, you'll know it after chatting with me more.
[人类]	你不是人工智能机器人么	[Human]	Aren't you an AI chatbot?
[小冰]	是又怎样，我不会冷冰冰的对 待你的	[XiaoIce]	So what? I won't be indifferent to you.
[人类]	你都有什么功能啊	[Human]	What functions do you have?
[小冰]	我有200多个skills，来，说出 你的需求	[XiaoIce]	I have over 200 skills. Just name one you need.

Figure 26.3 A sample Chinese conversation from the XiaoIce chatbot, together with translation from Zhou et al. (2018).

a previous conversation) or by using a machine translation paradigm such as neural network sequence-to-sequence systems, to learn to map from a user utterance to a system response.

26.2.1 Rule-based chatbots: ELIZA and PARRY

ELIZA (Weizenbaum, 1966) is the most important chatbot dialogue system in the history of the field, and so we take some time here to describe its algorithm in detail. ELIZA was designed to simulate a Rogerian psychologist, based on a branch of clinical psychology whose methods involve drawing the patient out by reflecting patient's statements back at them. Rogerian psychology is the rare type of conversation in which, as Weizenbaum points out, one can "assume the pose of knowing almost nothing of the real world". If a patient says "I went for a long boat ride" and the psychiatrist says "Tell me about boats", you don't assume she didn't know what a boat is, but rather assume she had some conversational goal. Most chatbots trying to pass the Turing test choose a domain with similar properties.

Fig. 26.4 shows the most famous ELIZA conversation, in which ELIZA seems

Men are all alike.
IN WHAT WAY
 They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
 Well my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
 He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
 ::
WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?
 My father
YOUR FATHER
 You are like my father in some ways.
WHAT RESEMBLANCE DO YOU SEE
 You are not very aggressive but I think you don't want me to notice that.
WHAT MAKES YOU THINK I AM NOT AGGRESSIVE
 You don't argue with me.
WHY DO YOU THINK I DON'T ARGUE WITH YOU
 You are afraid of me.
DOES IT PLEASE YOU TO BELIEVE I'M AFRAID OF YOU
 My father is afraid of everybody.
WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER
 Bullies.
DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE

Figure 26.4 Sample ELIZA dialogue from [Weizenbaum \(1966\)](#).

to simulate giving very sophisticated therapeutic advice, excerpted (deleting some material with ...) from a full interaction given in the original paper ([Weizenbaum, 1966](#)).

As we summarized briefly in Chapter 2, ELIZA worked by pattern/transform rules like the following one:

```
(0 YOU 0 ME) [pattern]
->
(WHAT MAKES YOU THINK I 3 YOU) [transform]
```

In the ELIZA pattern, 0 means Kleene*, and in the transform rules the numbers are the index of the constituent in the pattern. Thus the number 3 refers to the second 0 in the first pattern. This rule would transfer

You hate me
 into:
 WHAT MAKES YOU THINK I HATE YOU

Each ELIZA pattern/rule is linked to a keyword that might occur in a user sentence. The architecture is summarized in Fig. 26.5.

Keywords are associated with a rank, with specific words being more highly ranked, and more general words ranking lower. Consider the following user sentence:

```

function ELIZA GENERATOR(user sentence) returns response
  Find the word w in sentence that has the highest keyword rank
    if w exists
      Choose the highest ranked rule r for w that matches sentence
      response  $\leftarrow$  Apply the transform in r to sentence
      if w = ‘my’
        future  $\leftarrow$  Apply a transformation from the ‘memory’ rule list to sentence
        Push future onto memory stack
      else (no keyword applies)
        either
          response  $\leftarrow$  Apply the transform for the NONE keyword to sentence
        or
          response  $\leftarrow$  Pop the top response from the memory stack
    return(response)

```

Figure 26.5 A simplified sketch of the ELIZA algorithm. The power of the algorithm comes from the particular transforms associated with each keyword.

I know everybody laughed at me

Because it has the word “I”, this sentence could match the following rule whose keyword is *I*:

(I *) \rightarrow (You say you 2)

producing:

YOU SAY YOU KNOW EVERYBODY LAUGHED AT YOU

But as Weizenbaum points out, “I” is a very general word and its keywords lead to very general responses. Instead the keyword “everybody” is much more interesting, since someone using universals like everybody or always is probably “referring to some quite specific event or person”. Therefore, ELIZA prefers to respond with the pattern associated with the more specific keyword *everybody* (implementing by just assigning “everybody” rank 5 and “I” rank 0 in the lexicon), whose rule thus transforms the sentence to:

WHO IN PARTICULAR ARE YOU THINKING OF?

If no keyword matches, ELIZA chooses a non-committal response like “PLEASE GO ON”, “THAT’S VERY INTERESTING”, or “I SEE”.

Finally, ELIZA has a clever memory trick that accounts for the very last sentence of the conversation above. Whenever the word “my” is the highest ranked keyword, ELIZA will randomly select a transform on the MEMORY list, apply it to the sentence, and store it on the stack:

```

(MEMORY MY
  (0 MY 0 = LETS DISCUSS FURTHER WHY YOUR 3)
  (0 MY 0 = EARLIER YOU SAID YOUR 3)
  (0 MY 0 = DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR 3

```

Later, if no keyword matches a sentence, ELIZA will return the top of the MEMORY queue instead.²

² Fun fact: because of its structure as a queue, this MEMORY trick is the earliest known hierarchical model of discourse in natural language processing.

People became deeply emotionally involved with the program. Weizenbaum tells the story of one of his staff who would ask Weizenbaum to leave the room when she talked with ELIZA. When Weizenbaum suggested that he might want to store all the ELIZA conversations for later analysis, people immediately pointed out the privacy implications, which suggested that they were having quite private conversations with ELIZA, despite knowing that it was just software.

ELIZA's framework is still used today; modern chatbot system tools like ALICE are based on updated versions of ELIZA's pattern/action architecture.

A few years after ELIZA, another chatbot with a clinical psychology focus, PARRY ([Colby et al., 1971](#)), was used to study schizophrenia. In addition to ELIZA-like regular expressions, the PARRY system included a model of its own mental state, with affect variables for the agent's levels of fear and anger; certain topics of conversation might lead PARRY to become more angry or mistrustful. If PARRY's **anger** variable is high, he will choose from a set of "hostile" outputs. If the input mentions his delusion topic, he will increase the value of his **fear** variable and then begin to express the sequence of statements related to his delusion. Parry was the first known system to pass the Turing test (in 1972!); psychiatrists couldn't distinguish text transcripts of interviews with PARRY from transcripts of interviews with real paranoid ([Colby et al., 1972](#)).

26.2.2 Corpus-based chatbots

Corpus-based chatbots, instead of using hand-built rules, mine conversations of human-human conversations, (or sometimes mine the human sides of human-machine conversations).

These systems are enormously data-intensive; [Serban et al. \(2018\)](#) estimate that training modern chatbots require hundreds of millions or even billions of words. Many such corpora have been used, including large spoken conversational corpora like the Switchboard corpus of American English telephone conversations ([Godfrey et al., 1992](#)) or the various CALLHOME and CALLFRIEND telephone conversational corpora in many languages. Many systems also train on movie dialogue, which is available in great quantities in various corpora ([Lison and Tiedemann, 2016, inter alia](#)), and which resembles natural conversation in many ways ([Forchini, 2013](#)). Text from microblogging sites like Twitter ([Ritter et al., 2010](#)) or a Weibo (微博) have also been used, or datasets of crowdworker conversations like Topical-Chat ([Gopalakrishnan et al., 2019](#)). Many corpora also focus on specific topics, and can be used for topical chatbots. See [Serban et al. \(2018\)](#) for a comprehensive summary of available corpora. Another common technique is to extract possible responses from non-dialogue corpora, so that a chatbot can tell stories or mention facts acquired in that way.

Finally, once a chatbot has been put into practice, the turns that humans use to respond to the chatbot can be used as additional conversational data for training. The XiaoIce system collects and stores all human-machine conversations between XiaoIce and its users, resulting in a dataset of over 30 billion conversation pairs. It's crucial in these cases to remove personally identifiable information (PII); see Section 26.6.1.

The two main architectures for corpus-based chatbots: information retrieval, and machine learned sequence transduction. Like rule-based chatbots (but unlike frame-based dialogue systems), most corpus-based chatbots do very little modeling of the conversational context. Instead they tend to focus on generating a single response turn that is appropriate given the user's immediately previous utterance or two. For

response generation this reason they are often called **response generation** systems. Corpus-based chatbots thus have some similarity to question answering systems, which focus on single responses while ignoring context or larger conversational goals.

IR-based chatbots

The principle behind information retrieval based chatbots is to respond to a user's turn X by repeating some appropriate turn Y from a corpus of natural (human) text of the sort described in the prior section.

Given the corpus and the user's sentence, IR-based systems can use any retrieval algorithm to choose an appropriate response from the corpus. The two simplest methods are the following:

1. Return the response to the most similar turn: Given user query q and a conversational corpus C , find the turn t in C that is most similar to q (for example has the highest cosine with q) and return the *following* turn, i.e. the human response to t in C :

$$r = \text{response} \left(\underset{t \in C}{\operatorname{argmax}} \frac{q^T t}{\|q\| \|t\|} \right) \quad (26.1)$$

The idea is that we should look for a turn that most resembles the user's turn, and return the human response to that turn (Jafarpour et al. 2009, Leuski and Traum 2011).

2. Return the most similar turn: Given user query q and a conversational corpus C , return the turn t in C that is most similar to q (for example has the highest cosine with q):

$$r = \underset{t \in C}{\operatorname{argmax}} \frac{q^T t}{\|q\| \|t\|} \quad (26.2)$$

The idea here is to directly match the users query q with turns from C , since a good response will often share words or semantics with the prior turn.

In each case, any similarity function can be used, such as cosines computed either over words (weighted by tf-idf) or more commonly now, cosines over any kind of sentence embeddings.

Although returning the *response* to the most similar turn seems like a more intuitive algorithm, returning the most similar turn seems to work better in practice, perhaps because selecting the response adds another layer of indirection that can allow for more noise (Ritter et al. 2011, Wang et al. 2013).

The IR-based approach can be extended by using more features than just the words in the q . For example using the entire conversation with the user so far can be quite helpful when the user's query is short (like "Yes" or "OK"). Information about the user or sentiment or other information can also play a role. The IR-based approach can even draw responses from narrative (non-dialogue) text. The COBOT chatbot (Isbell et al., 2000) pioneered this approach, generating responses by selecting sentences from a corpus that combined the Unabomber Manifesto by Theodore Kaczynski, articles on alien abduction, the scripts of "The Big Lebowski" and "Planet of the Apes". Chatbots that want to generate informative turns such as answers to user questions can use texts like Wikipedia to draw on sentences that might contain those answers (Yan et al., 2016). XiaoIce similarly collects sentences from public lectures and news articles and searches them using IR based on query expansion from the user's turn to respond to turns like "Tell me something about Beijing" (Zhou et al., 2018).

Encoder decoder chatbots

“What ho!” I said.
 “What ho!” said Motty.
 “What ho! What ho!”
 “What ho! What ho! What ho!”
 After that it seemed rather difficult to go on with the conversation.”

Wodehouse My Man Jeeves

An alternate way to use a corpus to generate dialogue is to think of response generation as a task of *transducing* from the user’s prior turn to the system’s turn. This is basically the machine learning version of Eliza; the system learns from a corpus to transduce a question to an answer.

This idea was first developed by using phrase-based machine translation (Ritter et al., 2011) to translate a user turn to a system response. It quickly became clear, however, that the task of response generation was too different from machine translation. In machine translation, words or phrases in the source and target sentences tend to align well with each other, but in conversation, a user utterance may share no words or phrases with a coherent response.

Instead, (roughly contemporaneously by Shang et al. 2015, Vinyals and Le 2015, and Sordoni et al. 2015) transduction models for response generation were modeled instead using encoder-decoder models (Chapter 11), as shown in Fig. 26.6.

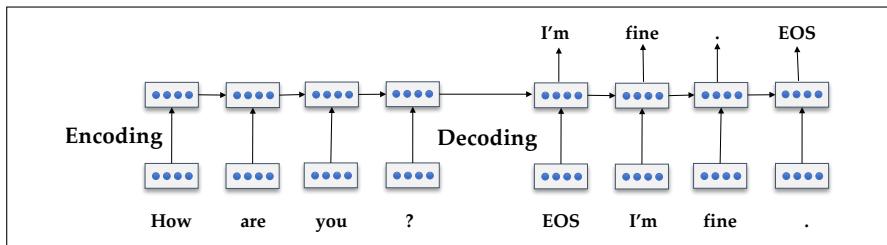


Figure 26.6 An encoder-decoder model for neural response generation in dialogue.

A number of modifications are required to the basic encoder-decoder model to adapt it for the task of response generation. For example basic encoder-decoder models have a tendency to produce predictable but repetitive and therefore dull responses like “I’m OK” or “I don’t know” that shut down the conversation. This can be addressed by changing the objective function for seq2seq model training to a mutual information objective, or by modifying a beam decoder to keep more diverse responses in the beam (Li et al., 2016).

Another problem with simple encoder-decoder response generation architecture is its inability to model the longer prior context of the conversation. This can be done by allowing the model to see prior turns, such as by using a hierarchical model that summarizes information over multiple prior turns (Lowe et al., 2017b).

Finally, encoder-decoder response generators focus on generating single responses, and so don’t tend to do a good job of continuously generating responses that cohere across multiple turns. This can be addressed by using reinforcement learning, as well as techniques like adversarial networks, to learn to choose responses that make the overall conversation more natural (Li et al. 2016, Li et al. 2017).

Fig. 26.7 shows some sample responses generated by a vanilla SEQ2SEQ model, and from a model trained by an adversarial algorithm to produce responses that are harder to distinguish from human responses (Li et al., 2017).

Input	tell me ... how long have you had this falling sickness ?
Vanilla-SEQ2SEQ	i'm not a doctor.
Adversarial	a few months, i guess .
Input	so i had the doctors test sammy 's response to conditioning .
Vanilla-SEQ2SEQ	sammy wrote the test sammy wrote the test .
Adversarial	so he took the pills .
Input	they didn 't have much success with this problem commander .
Vanilla-SEQ2SEQ	they 're not the only ones who have been in the system .
Adversarial	can we find someone else ?

Figure 26.7 Sample responses generated by a SEQ2SEQ model trained either with a vanilla maximum likelihood objective, or adversarially trained to produce sentences that are hard for an adversary to distinguish from human sentences (Li et al., 2017).

Evaluating Chatbots

Chatbots are generally evaluated by humans. The slot-filling evaluations used for task-based dialogue (Section 26.5) aren't appropriate for this task (Artstein et al., 2009), and word-overlap metrics like BLEU for comparing a chatbot's response to a human response turn out to correlate very poorly with human judgments (Liu et al., 2016). BLEU performs poorly because there are so many possible responses to any given turn; word-overlap metrics work best when the space of responses is small and lexically overlapping, as is the case in machine translation.

While human evaluation is therefore required for evaluating chatbots, there are beginning to be models for automatic evaluation. The ADEM (Lowe et al., 2017a) classifier is trained on a set of responses labeled by humans with how appropriate they are, and learns to predict this label from the dialogue context and the words in the system response.

adversarial evaluation

Another paradigm is **adversarial evaluation** (Bowman et al. 2016, Kannan and Vinyals 2016, Li et al. 2017), inspired by the Turing test. The idea is to train a “Turing-like” evaluator classifier to distinguish between human-generated responses and machine-generated responses. The more successful a response generation system is at fooling this evaluator, the better the system.

26.3 GUS: Simple Frame-based Dialogue Systems

GUS
dialogue state
frame
slot

We turn now to **task-based dialogue**, in which a dialogue system has the goal of helping a user solve some task like making an airplane reservation or buying a product. In this section we introduce the very simple **GUS** architecture for task-based dialogue. Although it is quite old, (it was first introduced in 1977 in the influential GUS system for travel planning (Bobrow et al., 1977)) the GUS architecture has been astonishingly long-lived and underlies most if not all modern commercial digital assistants. Figure 26.8 shows a conversation with the original GUS system.

All modern task-based dialogue systems, whether the simple GUS architecture we describe here, or the more sophisticated **dialogue state** architectures we turn to in the following section, are based around **frames**. A frame is a kind of knowledge structure representing the kinds of intentions the system can extract from user sentences, and consists of a collection of **slots**, each of which can take a set of possible **values**. Together this set of frames is sometimes called a **domain ontology**.

The set of slots in a task-based dialogue frame specifies what the system needs to know, and the filler of each slot is constrained to values of a particular semantic

GUS: Hello. My name is GUS. I can help you plan a simple trip by air. Where do you want to go ?
 Client: I want to go to San Diego on May 28
 GUS: What time do you want to leave?
 Client: I must be in San Diego before 10 am
 GUS: Would you like P.S.A. flight 102 that arrives at 9.15 am ?
 Client: What is the next flight ?
 GUS: Air California flight 310 that leaves at 8.30am and arrives at 10.00am
 Client: I'll take the first one
 GUS: In what name should I make a reservation, please?
 Client: Dan Bobrow
 GUS: I have confirmed the following flight: P.S.A. flight 102 on Wednesday May 28 from San Jose to San Diego that leaves at 7.30 am and arrives at 9.15 am.
 GUS: What date do you want to return on ?
 Client: On Friday in the evening.
 GUS: Would you like the flight that leaves at 7.45 pm ?
 Client: That's fine.
 GUS: I have confirmed the following flight: P.S.A. flight 307 on Friday May 30 from San Diego to San Jose that leaves at 7.45 pm and arrives at 9.30 pm Thank you for calling. Goodbye

Figure 26.8 The travel domain: A transcript of an actual dialogue with the GUS system of [Bobrow et al. \(1977\)](#). P.S.A. and Air California were airlines of that period.

type. In the travel domain, for example, a slot might be of type city (hence take on values like *San Francisco*, or *Hong Kong*) or of type date, airline, or time.

Slot	Type	Question Template
ORIGIN CITY	city	“From what city are you leaving?”
DESTINATION CITY	city	“Where are you going?”
DEPARTURE TIME	time	“When would you like to leave?”
DEPARTURE DATE	date	“What day would you like to leave?”
ARRIVAL TIME	time	“When do you want to arrive?”
ARRIVAL DATE	date	“What day would you like to arrive?”

Figure 26.9 A frame in a frame-based dialogue system, showing the type of each slot and a question used to fill the slot.

Types in GUS, as in modern frame-based dialogue agents, have hierarchical structure; for example the *date* type in GUS is itself a frame with slots with types like *integer* or members of sets of weekday names:

```
DATE
MONTH:NAME  YEAR:INTEGER  DAY:(BOUNDED-INTEGER 1 31)
WEEKDAY:(MEMBER (Sunday Monday Tuesday Wednesday
                  Thursday Friday Saturday))
```

26.3.1 Control structure for frame-based dialogue

The control architecture for frame-based dialogue systems, used in various forms in modern systems like Apple’s Siri, Amazon’s Alexa, and the Google Assistant, is designed around the frame. The system’s goal is to fill the slots in the frame with the fillers the user intends, and then perform the relevant action for the user (answering a question, or booking a flight).

To do this, the system asks questions of the user (using pre-specified question templates associated with each slot of each frame, as shown in Fig. 26.9), filling any

slot that the user specifies (we'll describe how slot-filling works in the next section). If a user's response fills multiple slots, like the following:

- (26.3) I want a flight from San Francisco to Denver one way leaving after five p.m. on Tuesday.

the system fills all the relevant slots, and then continues asking questions to fill the remaining slots, skipping questions associated with filled slots. The GUS architecture also has condition-action rules attached to slots. For example, a rule attached to the DESTINATION slot for the plane booking frame, once the user has specified the destination, might automatically enter that city as the default *StayLocation* for the related hotel booking frame. Or if the user specifies the DESTINATION DAY for a short trip the system could automatically enter the ARRIVAL DAY.

Many domains require multiple frames. Besides frames for car or hotel reservations, we might need frames with general route information (for questions like *Which airlines fly from Boston to San Francisco?*), or information about airfare practices (for questions like *Do I have to stay a specific number of days to get a decent airfare?*). The system must be able to disambiguate which slot of which frame a given input is supposed to fill and then switch dialogue control to that frame.

Because of this need to dynamically switch control, the GUS architecture is a **production rule** system. Different types of inputs cause different productions to fire, each of which can flexibly fill in different frames. The production rules can then switch control according to factors such as the user's input and some simple dialogue history like the last question that the system asked.

Once the system has enough information it performs the necessary action (like querying a database of flights) and returns the result to the user.

26.3.2 Natural language understanding for filling slots in GUS

intent determination

slot filling

The goal of the natural language understanding component in the frame-based architecture is to extract three things from the user's utterance. The first task is **domain classification**: is this user for example talking about airlines, programming an alarm clock, or dealing with their calendar? Of course this 1-of-n classification tasks is unnecessary for single-domain systems that are focused on, say, only calendar management, but multi-domain dialogue systems are the modern standard. The second is user **intent determination**: what general task or goal is the user trying to accomplish? For example the task could be to Find a Movie, or Show a Flight, or Remove a Calendar Appointment. Finally, we need to do **slot filling**: extract the particular slots and fillers that the user intends the system to understand from their utterance with respect to their intent. From a user utterance like this one:

Show me morning flights from Boston to San Francisco on Tuesday

a system might want to build a representation like:

DOMAIN:	AIR-TRAVEL
INTENT:	SHOW-FLIGHTS
ORIGIN-CITY:	Boston
ORIGIN-DATE:	Tuesday
ORIGIN-TIME:	morning
DEST-CITY:	San Francisco

while an utterance like

Wake me tomorrow at 6

should give an intent like this:

```
DOMAIN: ALARM-CLOCK
INTENT: SET-ALARM
TIME: 2017-07-01 0600-0800
```

The slot-filling method used in the original GUS system, and still quite common in industrial applications, is to use handwritten rules, often as part of the condition-action rules attached to slots or concepts. For example we might just define a regular expression for recognizing the SET-ALARM intent:

```
wake me (up) | set (the|an) alarm | get me up
```

semantic grammar

Rule-based research systems like the Phoenix system (Ward and Issar, 1994) consist of large hand-designed **semantic grammars** with thousands of rules. A semantic grammar is a context-free grammar in which the left-hand side of each rule corresponds to the semantic entities being expressed (i.e., the slot names) as in the following fragment:

SHOW	→ show me i want can i see ...
DEPART_TIME_RANGE	→ (after around before) HOUR morning afternoon evening
HOUR	→ one two three four... twelve (AMPM)
FLIGHTS	→ (a) flight flights
AMPM	→ am pm
ORIGIN	→ from CITY
DESTINATION	→ to CITY
CITY	→ Boston San Francisco Denver Washington

Semantic grammars can be parsed by any CFG parsing algorithm (see Chapter 13), resulting in a hierarchical labeling of the input string with semantic node labels, as shown in Fig. 26.10.



Figure 26.10 A semantic grammar parse for a user sentence, using slot names as the internal parse tree nodes.

It remains only to put the fillers into some sort of canonical form, for example by normalizing dates as discussed in Chapter 18.

Many industrial dialogue systems employ the GUS architecture but use supervised machine learning for slot-filling instead of these kinds of rules; see Section 26.4.2.

26.3.3 Other components of frame-based dialogue

The ASR (automatic speech recognition) component takes audio input from a phone or other device and outputs a transcribed string of words, as discussed in Chapter 28. The ASR component can also be made dependent on the dialogue state. For example, if the system has just asked the user “What city are you departing from?”, the

restrictive grammar**template-based generation****prompt**

ASR language model component can be constrained to assign high probability to answers with city names. This can be done by training a language model on answers to this question, or by hand-writing finite-state or context-free grammars for such sentences. A language model that is completely dependent on dialogue state is called a **restrictive grammar**; usually real systems mix restrictive grammars with more general language models.

The **natural language generation** module of any dialogue system produces the utterances that the system says to the user. Frame-based systems tend to use **template-based generation**, in which all or most of the words in the sentence to be uttered to the user are prespecified by the dialogue designer. Sentences created by these templates are often called **prompts**. Templates might be completely fixed (like ‘Hello, how can I help you?’), or can include some variables that are filled in by the generator, as in the following:

What time do you want to leave CITY-ORIG?
Will you return to CITY-ORIG from CITY-DEST?

It’s possible to do some simple grounding even in templated generation. Consider the unnaturalness of this example from [Cohen et al. \(2004\)](#):

(26.4) System: Did you want to review some more of your personal profile?
Caller: No.
System: What’s next?

Without an acknowledgment, the caller doesn’t know that the system has understood her ‘No’. The use of *Okay* below adds grounding to the templated response *What’s next?*, making (26.5) a much more natural response than (26.4):

(26.5) System: Did you want to review some more of your personal profile?
Caller: No.
System: *Okay*, what’s next?

The rule-based GUS approach is very common in industrial applications. As was true with the rule-based approach to information extraction, it has the advantage of high precision, and if the domain is narrow enough and experts are available, can provide sufficient coverage as well. On the other hand, the handwritten rules or grammars can be both expensive and slow to create, and handwritten rules can suffer from recall problems.

26.4 The Dialogue-State Architecture

Modern research systems for task-based dialogue are based on a more sophisticated version of the frame-based architecture called the **dialogue-state** or **belief-state** architecture. Figure 26.11 shows the six components of a typical dialogue-state system. The speech recognition and synthesis components deal with spoken language processing; we’ll return to them in Chapter 28.

For the rest of this chapter we therefore consider the other four components, which are part of both spoken and textual dialogue systems. These four components are more complex than in the simple GUS systems. For example, like the GUS systems, the dialogue-state architecture has an **NLU component** to extract slot fillers from the user’s utterance, but generally using machine learning rather than rules. The **dialogue state tracker** maintains the current state of the dialogue (which include the user’s most recent dialogue act, plus the entire set of slot-filler constraints the user

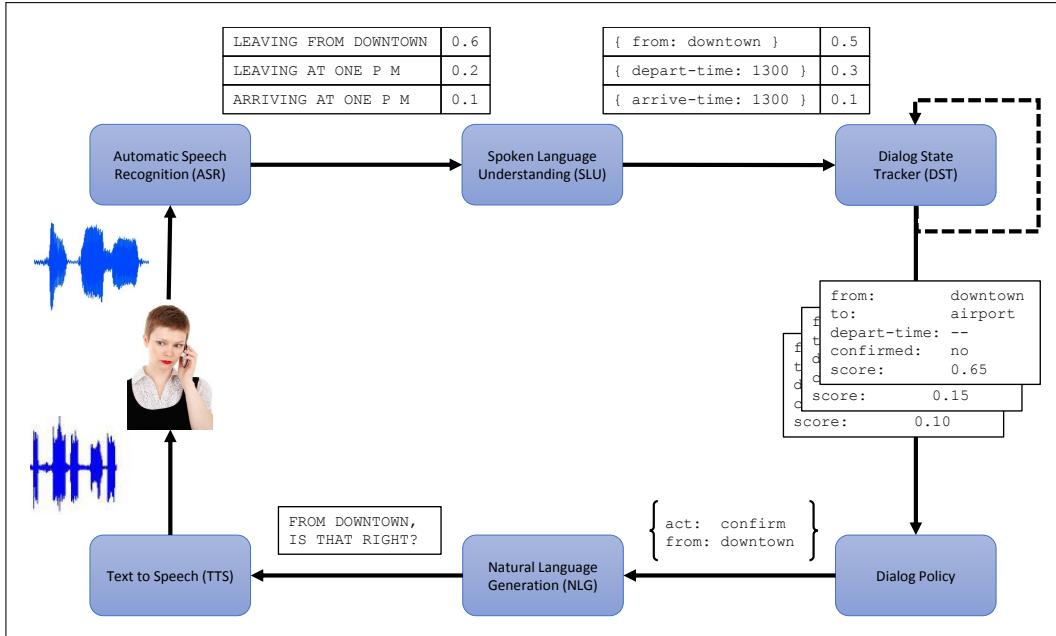


Figure 26.11 Architecture of a dialogue-state system for task-oriented dialogue from [Williams et al. \(2016\)](#).

has expressed so far). The **dialogue policy** decides what the system should do or say next. The dialogue policy in GUS was simple: ask questions until the frame was full and then report back the results of some database query. But a more sophisticated dialogue policy can help a system decide when to answer the user’s questions, when to instead ask the user a clarification question, when to make a suggestion, and so on. Finally, dialogue state systems have a **natural language generation** component. In GUS, the sentences that the generator produced were all from pre-written templates. But a more sophisticated generation component can condition on the exact context to produce turns that seem much more natural.

As of the time of this writing, most commercial system are architectural hybrids, based on GUS architecture augmented with some dialogue-state components, but there are a wide variety of dialogue-state systems being developed in research labs.

26.4.1 Dialogue Acts

dialogue acts

Dialogue-state systems make use of **dialogue acts**. Dialogue acts represent the interactive function of the turn or sentence, combining the idea of speech acts and grounding into a single representation. Different types of dialogue systems require labeling different kinds of acts, and so the tagset—defining what a dialogue act is exactly—tends to be designed for particular tasks.

Figure 26.12 shows a tagset for a restaurant recommendation system, and Fig. 26.13 shows these tags labeling a sample dialogue from the HIS system ([Young et al., 2010](#)). This example also shows the content of each dialogue acts, which are the slot fillers being communicated. So the user might INFORM the system that they want Italian food near a museum, or CONFIRM with the system that the price is reasonable.

Tag	Sys	User	Description
HELLO($a = x, b = y, \dots$)	✓	✓	Open a dialogue and give info $a = x, b = y, \dots$
INFORM($a = x, b = y, \dots$)	✓	✓	Give info $a = x, b = y, \dots$
REQUEST($a, b = x, \dots$)	✓	✓	Request value for a given $b = x, \dots$
REQALTS($a = x, \dots$)	✗	✓	Request alternative with $a = x, \dots$
CONFIRM($a = x, b = y, \dots$)	✓	✓	Explicitly confirm $a = x, b = y, \dots$
CONFREQ($a = x, \dots, d$)	✓	✗	Implicitly confirm $a = x, \dots$ and request value of d
SELECT($a = x, a = y$)	✓	✗	Implicitly confirm $a = x, \dots$ and request value of d
AFFIRM($a = x, b = y, \dots$)	✓	✓	Affirm and give further info $a = x, b = y, \dots$
NEGATE($a = x$)	✗	✓	Negate and give corrected value $a = x$
DENY($a = x$)	✗	✓	Deny that $a = x$
BYE()	✓	✓	Close a dialogue

Figure 26.12 Dialogue acts used by the HIS restaurant recommendation system of [Young et al. \(2010\)](#). The Sys and User columns indicate which acts are valid as system outputs and user inputs, respectively.

Utterance	Dialogue act
U: Hi, I am looking for somewhere to eat.	hello(task = find, type=restaurant)
S: You are looking for a restaurant. What type of food do you like?	confreq(type = restaurant, food)
U: I'd like an Italian somewhere near the museum.	inform(food = Italian, near=museum)
S: Roma is a nice Italian restaurant near the museum.	inform(name = "Roma", type = restaurant, food = Italian, near = museum)
U: Is it reasonably priced?	confirm(pricerange = moderate)
S: Yes, Roma is in the moderate price range.	affirm(name = "Roma", pricerange = moderate)
U: What is the phone number?	request(phone)
S: The number of Roma is 385456.	inform(name = "Roma", phone = "385456")
U: Ok, thank you goodbye.	bye()

Figure 26.13 A sample dialogue from the HIS System of [Young et al. \(2010\)](#) using the dialogue acts in Fig. 26.12.

26.4.2 Slot Filling

The task of slot-filling, and the simpler tasks of domain and intent classification, are special cases of the task of supervised semantic parsing discussed in Chapter 17, in which we have a training set that associates each sentence with the correct set of slots, domain, and intent.

A simple method is to train a sequence model to map from input words representation to slot fillers, domain and intent. For example given the sentence:

I want to fly to San Francisco on Monday afternoon please

we compute a sentence representation, for example by passing the sentence through a contextual embedding network like BERT. The resulting sentence representation can be passed through a feedforward layer and then a simple 1-of-N classifier to determine that the domain is AIRLINE and the intent is SHOWFLIGHT.

IOB Our training data is sentences paired with sequences of **IOB** labels:

```
0 0      0 0    B-DES I-DES      0 B-DEPTIME I-DEPTIME 0
I want to fly to San Francisco on Monday afternoon please
```

Recall from Chapter 18 that in IOB tagging we introduce a tag for the beginning (B) and inside (I) of each slot label, and one for tokens outside (O) any slot label. The number of tags is thus $2n + 1$ tags, where n is the number of slots.

Fig. 26.14 shows the architecture. The input is a series of words $w_1 \dots w_n$, which is passed through a contextual embedding model to get contextual word representations. This is followed by a feedforward layer and a softmax at each token position over possible IOB tags, with the output is a series of IOB tags $s_1 \dots s_n$. We can also combine the domain-classification and intent-extraction tasks with slot-filling simply by adding a domain concatenated with an intent as the desired output for the final EOS token.



Figure 26.14 A simple architecture for slot filling, mapping the words in the input through contextual embeddings like BERT to an output classifier layer (which can be linear or something more complex), followed by softmax to generate a series of IOB tags (and including a final state consisting of a domain concatenated with an intent).

Once the sequence labeler has tagged the user utterance, a filler string can be extracted for each slot from the tags (e.g., “San Francisco”), and these word strings can then be normalized to the correct form in the ontology (perhaps the airport code ‘SFO’). This normalization can take place by using homonym dictionaries (specifying, for example, that SF, SFO, and San Francisco are the same place).

In industrial contexts, machine learning-based systems for slot-filling are often bootstrapped from GUS-style rule-based systems in a semi-supervised learning manner. A rule-based system is first built for the domain, and a test-set is carefully labeled. As new user utterances come in, they are paired with the labeling provided by the rule-based system to create training tuples. A classifier can then be trained on these tuples, using the test-set to test the performance of the classifier against the rule-based system. Some heuristics can be used to eliminate errorful training tuples, with the goal of increasing precision. As sufficient training samples become available the resulting classifier can often outperform the original rule-based system (Suendermann et al., 2009), although rule-based systems may still remain higher-precision for dealing with complex cases like negation.

26.4.3 Dialogue State Tracking

The job of the dialogue-state tracker is to determine both the current state of the frame (the fillers of each slot), as well as the user’s most recent dialogue act. The dialogue-state thus includes more than just the slot-filters expressed in the current sentence; it includes the entire state of the frame at this point, summarizing all of the user’s constraints. The following example from Mrkšić et al. (2017) shows the required output of the dialogue state tracker after each turn:

```

User: I'm looking for a cheaper restaurant
      inform(price=cheap)
System: Sure. What kind - and where?
User: Thai food, somewhere downtown
      inform(price=cheap, food=Thai, area=centre)
System: The House serves cheap Thai food
User: Where is it?
      inform(price=cheap, food=Thai, area=centre); request(address)
System: The House is at 106 Regent Street

```

Since dialogue acts place some constraints on the slots and values, the tasks of dialogue-act detection and slot-filling are often performed jointly. Consider the task of determining that

I'd like Cantonese food near the Mission District

has the structure

inform(food=cantonese,area=mission).

Dialogue act interpretation—in this example choosing `inform` from the set of dialogue acts for this task—is done by supervised classification trained on hand-labeled dialog acts, predicting the dialogue act tag based on embeddings representing the current input sentence and the prior dialogue acts.

The simplest dialogue state tracker might just take the output of a slot-filling sequence-model (Section 26.4.2) after each sentence. Alternatively, a more complex model can make use of the reading-comprehension architectures from Chapter 25. For example the model of Gao et al. (2019) trains a classifier for each slot to decide whether its value is being changed in the current sentence or should be carried over from the previous sentences. If the slot value is being changed, a span-prediction model is used to predict the start and end of the span with the slot filler.

A special case: detecting correction acts

user correction acts

Some dialogue acts are important because of their implications for dialogue control. If a dialogue system misrecognizes or misunderstands an utterance, the user will generally correct the error by repeating or reformulating the utterance. Detecting these **user correction acts** is therefore quite important. Ironically, it turns out that corrections are actually *harder* to recognize than normal sentences! In fact, corrections in one early dialogue system (the TOOT system) had double the ASR word error rate of non-corrections (Swerts et al., 2000)! One reason for this is that speakers sometimes use a specific prosodic style for corrections called **hyperarticulation**, in which the utterance contains exaggerated energy, duration, or F0 contours, such as *I said BAL-TI-MORE, not Boston* (Wade et al. 1992, Levow 1998, Hirschberg et al. 2001). Even when they are not hyperarticulating, users who are frustrated seem to speak in a way that is harder for speech recognizers (Goldberg et al., 2003).

hyperarticulation

What are the characteristics of these corrections? User corrections tend to be either exact repetitions or repetitions with one or more words omitted, although they may also be paraphrases of the original utterance. (Swerts et al., 2000). Detecting these reformulations or correction acts can be part of the general dialogue act detection classifier. Alternatively, because the cues to these acts tend to appear in different ways than for simple acts (like `INFORM` or `request`, we can make use of features orthogonal to simple contextual embedding features; some typical features are shown below (Levow 1998, Litman et al. 1999, Hirschberg et al. 2001, Bulyko et al. 2005, Awadallah et al. 2015):

features	examples
lexical	words like “no”, “correction”, “I don’t”, or even swear words, utterance length
semantic	similarity (word overlap or embedding cosine) between the candidate correction act and the user’s prior utterance
phonetic	phonetic overlap between the candidate correction act and the user’s prior utterance (i.e. “WhatsApp” may be incorrectly recognized as “What’s up”)
prosodic	hyperarticulation, increases in F0 range, pause duration, and word duration, generally normalized by the values for previous sentences
ASR	ASR confidence, language model probability

26.4.4 Dialogue Policy

dialogue policy The goal of the **dialogue policy** is to decide what action the system should take next, that is, what dialogue act to generate.

More formally, at turn i in the conversation we want to predict which action A_i to take, based on the entire dialogue state. The state could mean the entire sequence of dialogue acts from the system (A) and from the user (U), in which case the task would be to compute:

$$\hat{A}_i = \operatorname{argmax}_{A_i \in A} P(A_i | (A_1, U_1, \dots, A_{i-1}, U_{i-1})) \quad (26.6)$$

We can simplify this by maintaining as the dialogue state mainly just the set of slot-fillers that the user has expressed, collapsing across the many different conversational paths that could lead to the same set of filled slots.

Such a policy might then just condition on the current dialogue state as represented just by the current state of the frame Frame_i (which slots are filled and with what) and the last turn by the system and user:

$$\hat{A}_i = \operatorname{argmax}_{A_i \in A} P(A_i | \text{Frame}_{i-1}, A_{i-1}, U_{i-1}) \quad (26.7)$$

These probabilities can be estimated by a neural classifier using neural representations of the slot fillers (for example as spans) and the utterances (for example as sentence embeddings computed over contextual embeddings)

More sophisticated models train the policy via **reinforcement learning**. To decide which action to take, a reinforcement learning system gets a reward at the end of the dialogue, and uses that reward to train a policy to take actions. For example in the movie-recommendation dialogue system of [Fazel-Zarandi et al. \(2017\)](#), the action space has only three actions: EXECUTE, CONFIRM, and ELICIT. The EXECUTE sends a query to the database and answers the user’s question, CONFIRM clarifies the intent or slot with the users (e.g., “Do you want movies directed by Christopher Nolan?”) while ELICIT asks the user for missing information (e.g., “Which movie are you talking about?”). The system gets a large positive reward if the dialogue system terminates with the correct slot representation at the end, a large negative reward if the slots are wrong, and a small negative reward for confirmation and elicitation questions to keep the system from re-confirming everything.

Policy Example: Confirmation and Rejection

Modern dialogue systems often make mistakes. It is therefore important for dialogue systems to make sure that they have achieved the correct interpretation of the user’s

explicit confirmation

input. This is generally done by two methods: **confirming** understandings with the user and **rejecting** utterances that the system is likely to have misunderstood.

As we saw in the prior section, most systems introduce particular strategies and actions related to confirmation and rejection. When using the **explicit confirmation** strategy, a system asks the user a direct question to confirm the system's understanding, like the two examples below in which the system asks a (boldface) yes-no confirmation questions:

S: Which city do you want to leave from?
U: Baltimore.
S: Do you want to leave from Baltimore?
U: Yes.
U: I'd like to fly from Denver Colorado to New York City on September twenty first in the morning on United Airlines
S: Let's see then. I have you going from Denver Colorado to New York on September twenty first. Is that correct?
U: Yes

implicit confirmation

When using the **implicit confirmation** strategy, a system instead can demonstrate its understanding as a **grounding** strategy, for example repeating back the system's understanding as part of asking the next question, as in the two examples below:

U: I want to travel to Berlin
S: When do you want to travel to Berlin?
U2: Hi I'd like to fly to Seattle Tuesday Morning
A3: Traveling to Seattle on Tuesday, August eleventh in the morning.
Your full name?

rejection

Explicit and implicit confirmation have complementary strengths. Explicit confirmation makes it easier for users to correct the system's misrecognition since a user can just answer “no” to the confirmation question. But explicit confirmation is awkward and increases the length of the conversation (Danieli and Gerbino 1995, Walker et al. 1998). The explicit confirmation dialogue fragments above sound non-natural and definitely non-human; implicit confirmation is much more conversationally natural.

Confirmation is just one kind of conversational action by which a system can express lack of understanding. Another option is **rejection**, in which a system gives the user a prompt like *I'm sorry, I didn't understand that*.

progressive prompting

Sometimes utterances are rejected multiple times. This might mean that the user is using language that the system is unable to follow. Thus, when an utterance is rejected, systems often follow a strategy of **progressive prompting** or **escalating detail** (Yankelovich et al. 1995, Weinschenk and Barker 2000), as in this example from Cohen et al. (2004):

System: When would you like to leave?
Caller: Well, um, I need to be in New York in time for the first World Series game.
System: <reject>. Sorry, I didn't get that. Please say the month and day you'd like to leave.
Caller: I wanna go on October fifteenth.

In this example, instead of just repeating “When would you like to leave?”, the rejection prompt gives the caller more guidance about how to formulate an utterance the system will understand. These *you-can-say* help messages are important in helping improve systems’ understanding performance (Bohus and Rudnicky, 2005). If the caller’s utterance gets rejected yet again, the prompt can reflect this (“I still didn’t get that”), and give the caller even more guidance.

rapid reprompting

An alternative strategy for error handling is **rapid reprompting**, in which the system rejects an utterance just by saying “I’m sorry?” or “What was that?” Only if the caller’s utterance is rejected a second time does the system start applying progressive prompting. Cohen et al. (2004) summarize experiments showing that users greatly prefer rapid reprompting as a first-level error prompt.

It is common to use rich features other than just the dialogue state representation to make policy decisions. For example, the **confidence** that the ASR system assigns to an utterance can be used by explicitly confirming low-confidence sentences. As we’ll see in page ??, confidence is a metric that the speech recognizer can assign to its transcription of a sentence to indicate how confident it is in that transcription. Confidence is often computed from the acoustic log-likelihood of the utterance (greater probability means higher confidence), but prosodic features can also be used in confidence prediction. For example, utterances with large F0 excursions or longer durations, or those preceded by longer pauses, are likely to be misrecognized (Litman et al., 2000).

Another common feature in confirmation is the **cost** of making an error. For example, explicit confirmation is common before a flight is actually booked or money in an account is moved. Systems might have a four-tiered level of confidence with three thresholds α , β , and γ :

$< \alpha$	low confidence	reject
$\geq \alpha$	above the threshold	confirm explicitly
$\geq \beta$	high confidence	confirm implicitly
$\geq \gamma$	very high confidence	don’t confirm at all

26.4.5 Natural language generation in the dialogue-state model

Finally, once the policy has decided what speech act to generate, the natural language generation component needs to generate the text of a response to the user.

content planning sentence realization

Once a dialogue act has been decided, we need to generate the text of the response to the user. The task of natural language generation (NLG) in the information-state architecture is often modeled in two stages, **content planning** (what to say), and **sentence realization** (how to say it).

Here we’ll assume content planning has been done by the dialogue policy, which has chosen the dialogue act to generate, and chosen some attributes (slots and values) that the planner wants to say to the user (either to give the user the answer, or as part of a confirmation strategy).

Fig. 26.15 shows some sample input/outputs for the sentence realization phase. In the first example, the content planner has chosen the dialogue act RECOMMEND and some particular slots (name, neighborhood, cuisine) and their fillers. The goal of the sentence realizer is to generate a sentence like lines 1 or 2 shown in the figure, by training on many such examples of representation/sentence pairs from a large corpus of labeled dialogues.

Training data is hard to come by; we are unlikely to see every possible restaurants with every possible attribute in many possible differently worded sentences. There-

```

recommend(restaurant name= Au Midi, neighborhood = midtown,
cuisine = french)
1 Au Midi is in Midtown and serves French food.
2 There is a French restaurant in Midtown called Au Midi.
recommend(restaurant name= Loch Fyne, neighborhood = city
centre, cuisine = seafood)
3 Loch Fyne is in the City Center and serves seafood food.
4 There is a seafood restaurant in the City Centre called Loch Fyne.

```

Figure 26.15 Two examples of inputs to the sentence realization phase of NLG, showing the dialogue act and attributes prespecified by the content planner. Line 1-2 and 3-4 show different possible output sentences to be generated by the sentence realizer. From the restaurant recommendation system of Nayak et al. (2017).

fore it is common in sentence realization to increase the generality of the training examples by **delexicalization**. Delexicalization is the process of replacing specific words in the training set that represent slot values with a generic placeholder token representing the slot. Fig. 26.16 shows the result of delexicalizing the training sentences in Fig. 26.15.

```

recommend(restaurant name= Au Midi, neighborhood = midtown,
cuisine = french)
1 restaurant.name is in neighborhood and serves cuisine food.
2 There is a cuisine restaurant in neighborhood called restaurant.name.

```

Figure 26.16 Delexicalized sentences that can be used for generating many different relexicalized sentences. From the restaurant recommendation system of Nayak et al. (2017).

Mapping from frames to delexicalized sentences is generally done by encoder decoder models (Wen et al. 2015a, Wen et al. 2015b, Mrkšić et al. 2017, *inter alia*), trained on large hand-labeled corpora of task-oriented dialogue (Budzianowski et al., 2018). The input to the encoder is a sequence of tokens x_t that represent the dialogue act and its arguments. Thus the attribute/value pairs `decor:decent`, `service:good`, `cuisine: null` might be represented as a flat sequence of tokens, each mapped to a learned embedding w_t , as shown in Fig. 26.17.

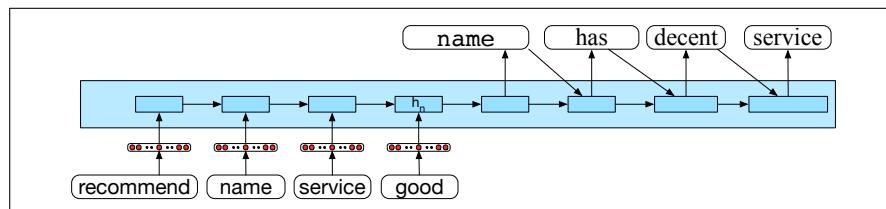


Figure 26.17 An encoder-decoder sentence realizer mapping slots/fillers to English.

The encoder reads all the input slot/value representations, produces a context vector that is used as input to the lexical decoder, which generates an English sentence. Let's suppose in this case we produce the following (delexicalized) sentence:

restaurant.name has decent service

Then once we've generated the delexicalized string, we can use the input frame from the content planner to **relexicalize** (fill in the exact restaurant or neighborhood or cuisine). This sentence is then relexicalized from the true values in the input frame, resulting in the final sentence:

Au Midi has decent service

TTS Performance	Was the system easy to understand ?
ASR Performance	Did the system understand what you said?
Task Ease	Was it easy to find the message/flight/train you wanted?
Interaction Pace	Was the pace of interaction with the system appropriate?
User Expertise	Did you know what you could say at each point?
System Response	How often was the system sluggish and slow to reply to you?
Expected Behavior	Did the system work the way you expected it to?
Future Use	Do you think you'd use the system in the future?

Figure 26.18 User satisfaction survey, adapted from [Walker et al. \(2001\)](#).

Generating Clarification Questions

clarification questions

It's also possible to design NLG algorithms that are specific to a particular dialogue act. For example, consider the task of generating **clarification questions**, in cases where the speech recognition fails to understand some part of the user's utterance. While it is possible to use the generic dialogue act REJECT ("Please repeat", or "I don't understand what you said"), studies of human conversations show that humans instead use targeted clarification questions that reprise elements of the misunderstanding ([Purver 2004](#), [Ginzburg and Sag 2000](#), [Stoyanov et al. 2013](#)).

For example, in the following hypothetical example the system reprises the words "going" and "on the 5th" to make it clear which aspect of the user's turn the system needs to be clarified:

User: What do you have going to UNKNOWN_WORD on the 5th?

System: Going where on the 5th?

Targeted clarification questions can be created by rules (such as replacing "going to UNKNOWN_WORD" with "going where") or by building classifiers to guess which slots might have been misrecognized in the sentence ([Chu-Carroll and Carpenter 1999](#), [Stoyanov et al. 2014](#), [Stoyanov and Johnston 2015](#)).

26.5 Evaluating Dialogue Systems

Evaluation is crucial in dialogue system design. If the task is unambiguous, we can simply measure absolute task success (did the system book the right plane flight, or put the right event on the calendar).

To get a more fine-grained idea of user happiness, we can compute a *user satisfaction rating*, having users interact with a dialogue system to perform a task and then having them complete a questionnaire. For example, Fig. 26.18 shows sample multiple-choice questions ([Walker et al., 2001](#)); responses are mapped into the range of 1 to 5, and then averaged over all questions to get a total user satisfaction rating.

It is often economically infeasible to run complete user satisfaction studies after every change in a system. For this reason, it is useful to have performance evaluation heuristics that correlate well with human satisfaction. A number of such factors and heuristics have been studied, often grouped into two kinds of criteria: how well the system allows users to accomplish their goals (maximizing task success) with the fewest problems (minimizing costs):

Task completion success:

Task success can be measured by evaluating the correctness of the total solution. For a frame-based architecture, this might be **slot error rate** the percentage of slots that

were filled with the correct values:

$$\text{Slot Error Rate for a Sentence} = \frac{\# \text{ of inserted/deleted/substituted slots}}{\# \text{ of total reference slots for sentence}} \quad (26.8)$$

For example consider a system given this sentence:

(26.9) Make an appointment with Chris at 10:30 in Gates 104

which extracted the following candidate slot structure:

Slot	Filler
PERSON	Chris
TIME	11:30 a.m.
ROOM	Gates 104

Here the slot error rate is 1/3, since the TIME is wrong. Instead of error rate, slot precision, recall, and F-score can also be used.

Interestingly, sometimes the user's *perception* of whether they completed the task is a better predictor of user satisfaction than the actual task completion success. (Walker et al., 2001).

A perhaps more important, although less fine-grained, measure of success is an extrinsic metric like **task error rate**. In this case, the task error rate would quantify how often the correct meeting was added to the calendar at the end of the interaction.

Efficiency cost:

Efficiency costs are measures of the system's efficiency at helping users. This can be measured by the total elapsed time for the dialogue in seconds, the number of total turns or of system turns, or the total number of queries (Polifroni et al., 1992). Other metrics include the number of system non-responses and the "turn correction ratio": the number of system or user turns that were used solely to correct errors divided by the total number of turns (Danieli and Gerbino 1995, Hirschman and Pao 1993).

Quality cost:

Quality cost measures other aspects of the interactions that affect users' perception of the system. One such measure is the number of times the ASR system failed to return any sentence, or the number of ASR rejection prompts. Similar metrics include the number of times the user had to barge in (interrupt the system), or the number of time-out prompts played when the user didn't respond quickly enough. Other quality metrics focus on how well the system understood and responded to the user. The most important is the **slot error rate** described above, but other components include the inappropriateness (verbose or ambiguous) of the system's questions, answers, and error messages or the correctness of each question, answer, or error message (Zue et al. 1989, Polifroni et al. 1992).

26.6 Dialogue System Design

The user plays a more important role in dialogue systems than in most other areas of speech and language processing, and thus the study of dialogue systems is closely linked with the field of Human-Computer Interaction (HCI). The design of dialogue strategies, prompts, and error messages, is often called **voice user interface** design, and generally follows **user-centered design** principles (Gould and Lewis, 1985):

Wizard-of-Oz system

1. Study the user and task: Understand the potential users and the nature of the task by interviews with users, investigation of similar systems, and study of related human-human dialogues.

2. Build simulations and prototypes: A crucial tool in building dialogue systems is the **Wizard-of-Oz system**. In wizard systems, the users interact with what they think is a software agent but is in fact a human “wizard” disguised by a software interface (Gould et al. 1983, Good et al. 1984, Fraser and Gilbert 1991). The name comes from the children’s book *The Wizard of Oz* (Baum, 1900), in which the wizard turned out to be just a simulation controlled by a man behind a curtain or screen.

A Wizard-of-Oz system can be used to test out an architecture before implementation; only the interface software and databases need to be in place. The wizard gets input from the user, has a graphical interface to a database to run sample queries based on the user utterance, and then has a way to output sentences, either by typing them or by some combination of selecting from a menu and typing. The wizard’s linguistic output can be disguised by a text-to-speech system or, more frequently, by using text-only interactions.

The results of a Wizard-of-Oz system can also be used as training data to train a pilot dialogue system. While Wizard-of-Oz systems are very commonly used, they are not a perfect simulation; it is difficult for the wizard to exactly simulate the errors, limitations, or time constraints of a real system; results of wizard studies are thus somewhat idealized, but still can provide a useful first idea of the domain issues.



3. Iteratively test the design on users: An iterative design cycle with embedded user testing is essential in system design (Nielsen 1992, Cole et al. 1997, Yankelovich et al. 1995, Landauer 1995). For example in a famous anecdote in dialogue design history, an early dialogue system required the user to press a key to interrupt the system Stifelman et al. (1993). But user testing showed users barged in, which led to a redesign of the system to recognize overlapped speech. The iterative method is also important for designing prompts that cause the user to respond in normative ways.

There are a number of good books on conversational interface design (Cohen et al. 2004, Harris 2005, Pearl 2017).

26.6.1 Ethical Issues in Dialogue System Design

Ethical issues have long been understood to be crucial in the design of artificial agents, predating the conversational agent itself. Mary Shelley’s classic discussion of the problems of creating agents without a consideration of ethical and humanistic concerns lies at the heart of her novel *Frankenstein*. One important ethical issue has to do with bias. As we discussed in Section 6.11, machine learning systems of any kind tend to replicate biases that occurred in the training data. This is especially relevant for chatbots, since



both IR-based and neural transduction architectures are designed to respond by approximating the responses in the training data.

A well-publicized instance of this occurred with Microsoft's 2016 **Tay** chatbot, which was taken offline 16 hours after it went live, when it began posting messages with racial slurs, conspiracy theories, and personal attacks. Tay had learned these biases and actions from its training data, including from users who seemed to be purposely teaching it to repeat this kind of language (Neff and Nagy, 2016).

Henderson et al. (2017) examined some standard dialogue datasets (drawn from Twitter, Reddit, or movie dialogues) used to train corpus-based chatbots, measuring bias (Hutto et al., 2015) and offensive and hate speech (Davidson et al., 2017). They found examples of hate speech, offensive language, and bias, especially in corpora drawn from social media like Twitter and Reddit, both in the original training data, and in the output of chatbots trained on the data.

Another important ethical issue is privacy. Already in the first days of ELIZA, Weizenbaum pointed out the privacy implications of people's revelations to the chatbot. Henderson et al. (2017) point out that home dialogue agents may accidentally record a user revealing private information (e.g. "Computer, turn on the lights –answers the phone –Hi, yes, my password is..."), which may then be used to train a conversational model. They showed that when an encoder-decoder dialogue model is trained on a standard corpus augmented with training keypairs representing private data (e.g. the keyphrase "social security number" followed by a number), an adversary who gave the keyphrase was able to recover the secret information with nearly 100% accuracy. Chatbots that are trained on transcripts of human-human or human-machine conversation must therefore anonymize personally identifiable information. It is the role of the Institutional Review Board (**IRB**) at a researcher's institution to review research proposals for such ethical issues.

Finally, chatbots raise important issues of gender equality. Current chatbots are overwhelmingly given female names, likely perpetuating the stereotype of a subservient female servant (Paolino, 2017). And when users use sexually harassing language, most commercial chatbots evade or give positive responses rather than responding in clear negative ways (Fessler, 2017).

26.7 Summary

Conversational agents are crucial speech and language processing applications that are already widely used commercially.

- In human dialogue, speaking is a kind of action; these acts are referred to as speech acts or dialogue acts. Speakers also attempt to achieve **common ground** by acknowledging that they have understand each other. Conversation also is characterized by turn structure and dialogue structure.
- Chatbots are conversational agents designed to mimic the appearance of informal human conversation. Rule-based chatbots like ELIZA and its modern descendants use rules to map user sentences into system responses. Corpus-based chatbots mine logs of human conversation to learn to automatically map user sentences into system responses.
- For task-based dialogue, most commercial dialogue systems use the GUS or

frame-based architecture, in which the designer specifies frames consisting of slots that the system must fill by asking the user.

- The **dialogue-state** architecture augments the GUS frame-and-slot architecture with richer representations and more sophisticated algorithms for keeping track of user's dialogue acts, **policies** for generating its own dialogue acts, and a natural language component.
- Dialogue systems are a kind of human-computer interaction, and general HCI principles apply in their design, including the role of the user, simulations such as Wizard-of-Oz systems, and the importance of iterative design and testing on real users.

Bibliographical and Historical Notes

The earliest conversational systems were chatbots like ELIZA ([Weizenbaum, 1966](#)) and PARRY ([Colby et al., 1971](#)). ELIZA had a widespread influence on popular perceptions of artificial intelligence, and brought up some of the first ethical questions in natural language processing —such as the issues of privacy we discussed above as well the role of algorithms in decision-making— leading its creator Joseph Weizenbaum to fight for social responsibility in AI and computer science in general.

Another early system, the GUS system ([Bobrow et al., 1977](#)) had by the late 1970s established the main frame-based paradigm that became the dominant industrial paradigm for dialogue systems for over 30 years.

In the 1990s, stochastic models that had first been applied to natural language understanding began to be applied to dialogue slot filling ([Miller et al. 1994](#), [Pieraccini et al. 1991](#)).

By around 2010 the GUS architecture finally began to be widely used commercially in phone-based dialogue systems like Apple's SIRI ([Bellegarda, 2013](#)) and other digital assistants.

The rise of the web and online chatbots brought new interest in chatbots and gave rise to corpus-based chatbot architectures around the turn of the century, first using information retrieval models and then in the 2010s, after the rise of deep learning, with sequence-to-sequence models.

The idea that utterances in a conversation are a kind of **action** being performed by the speaker was due originally to the philosopher [Wittgenstein \(1953\)](#) but worked out more fully by [Austin \(1962\)](#) and his student John Searle. Various sets of speech acts have been defined over the years, and a rich linguistic and philosophical literature developed, especially focused on explaining the use of indirect speech acts.

The idea of dialogue acts draws also from a number of other sources, including the ideas of adjacency pairs, pre-sequences, and other aspects of the international properties of human conversation developed in the field of **conversation analysis** (see [Levinson \(1983\)](#) for an introduction to the field).

This idea that acts set up strong local dialogue expectations was also prefigured by [Firth \(1935, p. 70\)](#), in a famous quotation:

Most of the give-and-take of conversation in our everyday life is stereotyped and very narrowly conditioned by our particular type of culture. It is a sort of roughly prescribed social ritual, in which you generally say what the other fellow expects you, one way or the other, to say.

Another important research thread modeled dialogue as a kind of collaborative behavior, including the ideas of common ground (Clark and Marshall, 1981), reference as a collaborative process (Clark and Wilkes-Gibbs, 1986), joint intention (Levesque et al., 1990), and shared plans (Grosz and Sidner, 1980).

The dialogue-state model was also strongly informed by analytic work on the linguistic properties of dialogue acts and on methods for their detection (Sag and Liberman 1975, Hinkelman and Allen 1989, Nagata and Morimoto 1994, Goodwin 1996, Chu-Carroll 1998, Shriberg et al. 1998, Stolcke et al. 2000, Gravano et al. 2012).

BDI

Two important lines of research that we were unable to cover in the chapter focused on the computational properties of conversational structure. One line, first suggested by Bruce (1975), suggested that since speech acts are actions, they should be planned like other actions, and drew on the AI planning literature (Fikes and Nilsson, 1971). An agent seeking to find out some information can come up with the plan of asking the interlocutor for the information. An agent hearing an utterance can interpret a speech act by running the planner “in reverse”, using inference rules to infer from what the interlocutor said what the plan might have been. Plan-based models of dialogue are referred to as **BDI** models because such planners model the **beliefs**, **desires**, and **intentions** (BDI) of the agent and interlocutor. BDI models of dialogue were first introduced by Allen, Cohen, Perrault, and their colleagues in a number of influential papers showing how speech acts could be generated (Cohen and Perrault, 1979) and interpreted (Perrault and Allen 1980, Allen and Perrault 1980). At the same time, Wilensky (1983) introduced plan-based models of understanding as part of the task of interpreting stories.

Another influential line of research focused on modeling the hierarchical structure of dialogue. Grosz’s pioneering (1977b) dissertation first showed that “task-oriented dialogues have a structure that closely parallels the structure of the task being performed” (p. 27), leading to her work with Sidner and others showing how to use similar notions of intention and plans to model discourse structure and coherence in dialogue. See, e.g., Lochbaum et al. (2000) for a summary of the role of intentional structure in dialogue.

The idea of applying reinforcement learning to dialogue first came out of AT&T and Bell Laboratories around the turn of the century with work on MDP dialogue systems (Walker 2000, Levin et al. 2000, Singh et al. 2002) and work on cue phrases, prosody, and rejection and confirmation. Reinforcement learning research turned quickly to the more sophisticated POMDP models (Roy et al. 2000, Lemon et al. 2006, Williams and Young 2007) applied to small slot-filling dialogue tasks,

Affect has played an important role in dialogue systems since its earliest days. In more recent work Mairesse and Walker (2008) showed that conversational agents are received better by users if they match users’ personality expectations. Rashkin et al. (2019) introduced the EMPATHETICDIALOGUES dataset of 25k conversations grounded in emotional situations, and (Lin et al., 2019) used mixtures of empathetic listeners (MoEL), each optimized to react to particular emotions, to generate empathetic responses.

[TBD: History of deep reinforcement learning here.] [TBD: surveys: Tur and De Mori (2011), Gao et al. (2019)]

[TBD: more history here on dialogue state tracking, NLG, end-to-end neural systems, etc]

Exercises

- dispreferred response**
- 26.1** Write a finite-state automaton for a dialogue manager for checking your bank balance and withdrawing money at an automated teller machine.
 - 26.2** A **dispreferred response** is a response that has the potential to make a person uncomfortable or embarrassed in the conversational context; the most common example dispreferred responses is turning down a request. People signal their discomfort with having to say no with surface cues (like the word *well*), or via significant silence. Try to notice the next time you or someone else utters a dispreferred response, and write down the utterance. What are some other cues in the response that a system might use to detect a dispreferred response? Consider non-verbal cues like eye gaze and body gestures.
 - 26.3** When asked a question to which they aren't sure they know the answer, people display their lack of confidence by cues that resemble other dispreferred responses. Try to notice some unsure answers to questions. What are some of the cues? If you have trouble doing this, read [Smith and Clark \(1993\)](#) and listen specifically for the cues they mention.
 - 26.4** Implement a small air-travel help system based on text input. Your system should get constraints from users about a particular flight that they want to take, expressed in natural language, and display possible flights on a screen. Make simplifying assumptions. You may build in a simple flight database or you may use a flight information system on the Web as your backend.
 - 26.5** Test your email-reading system on some potential users. Choose some of the metrics described in Section [26.5](#) and evaluate your system.

The characters or letters that are the basis of all the text-based methods we've seen so far in this book aren't just random symbols. They are also an amazing scientific invention: a theoretical model of the elements that make up human speech.

The earliest independently invented writing systems (Sumerian, Chinese, Mayan) were mainly logographic, which means one symbol representing a whole word. But from the earliest stages we can find, some of the symbols also represent the sounds that make up the words. Thus, the cuneiform sign to the right pronounced *ba* and meaning "ration" in Sumerian could also function purely as the sound /ba/ in languages that used cuneiform. Chinese writing, from its early instantiations on oracle bones, also assigns phonetic meaning to many character elements. Purely sound-based writing systems, whether syllabic (like Japanese *hiragana* or *katakana*), alphabetic (like the Roman alphabet used in this book), or consonantal (like Semitic writing systems), can generally be traced back to these early logo-syllabic systems, often as two cultures came together. Thus, the Arabic, Aramaic, Hebrew, Greek, and Roman systems all derive from a West Semitic script that is presumed to have been modified by Western Semitic mercenaries from a cursive form of Egyptian hieroglyphs. The Japanese syllabaries were modified from a cursive form of a set of Chinese characters that represented sounds. These Chinese characters themselves were used in Chinese to phonetically represent the Sanskrit in the Buddhist scriptures that were brought to China in the Tang dynasty.



Whatever its origins, the idea implicit in a sound-based writing system—that the spoken word is composed of smaller units of speech—underlies the modern algorithms for **speech recognition** (transcribing acoustic waveforms into strings of text words) and **speech synthesis** or **text-to-speech** (converting strings of text words into acoustic waveforms).

phonetics

In this chapter we introduce **phonetics** from a computational perspective. Phonetics is the study of the speech sounds used in the languages of the world, how they are produced by the articulators of the human vocal tract, how they are realized acoustically, and how this acoustic realization can be digitized and processed.

27.1 Speech Sounds and Phonetic Transcription

phone

Although a letter like 'p' or 'a' is a useful rough model of the sounds of human speech, in speech processing we often model the pronunciation of a word instead as a string of **phones**. A phone is a speech sound, represented with phonetic symbols modeled on letters in the Roman alphabet.

ARPAbet Symbol	IPA Symbol	Word	ARPAbet Transcription
[p]	[p]	parsley	[p aa r s l iy]
[t]	[t]	tea	[t iy]
[k]	[k]	cook	[k uh k]
[b]	[b]	bay	[b ey]
[d]	[d]	dill	[d ih l]
[g]	[g]	garlic	[g aa r l ix k]
[m]	[m]	mint	[m ih n t]
[n]	[n]	nutmeg	[n ah t m eh g]
[ng]	[ŋ]	baking	[b ey k ix ng]
[f]	[f]	flour	[f l aw axr]
[v]	[v]	clove	[k l ow v]
[th]	[θ]	thick	[th ih k]
[dh]	[ð]	those	[dh ow z]
[s]	[s]	soup	[s uw p]
[z]	[z]	eggs	[eh g z]
[sh]	[ʃ]	squash	[s k w aa sh]
[zh]	[ʒ]	ambrosia	[ae m b r ow zh ax]
[ch]	[tʃ]	cherry	[ch eh r iy]
[jh]	[dʒ]	jar	[jh aa r]
[l]	[l]	licorice	[l ih k axr ix sh]
[w]	[w]	kiwi	[k iy w iy]
[r]	[r]	rice	[r ay s]
[y]	[j]	yellow	[y eh l ow]
[h]	[h]	honey	[h ah n iy]

Figure 27.1 ARPAbet symbols for transcribing English consonants, with IPA equivalents.

This section surveys the different phones of English, focusing on American English. The **International Phonetic Alphabet (IPA)** is an evolving standard originally developed by the International Phonetic Association in 1888 with the goal of transcribing the sounds of all human languages. The ARPAbet (Shoup, 1980) is a phonetic alphabet designed for American English that uses ASCII symbols; it can be thought of as a convenient ASCII representation of an American-English subset of the IPA. Because the ARPAbet is common for computational modeling, we rely on it here. Figures 27.1 and 27.2 show the ARPAbet symbols for transcribing consonants and vowels, respectively, together with their IPA equivalents.

Many of the IPA and ARPAbet symbols are equivalent to familiar Roman letters. So, for example, the ARPAbet phone [p] represents the consonant sound at the beginning of *platypus*, *puma*, and *plantain*, the middle of *leopard*, or the end of *antelope*. In general, however, the mapping between the letters of English orthography and phones is relatively **opaque**; a single letter can represent very different sounds in different contexts. The English letter *c* corresponds to phone [k] in *cougar* [k uw g axr], but phone [s] in *cell* [s eh l]. Besides appearing as *c* and *k*, the phone [k] can appear as part of *x* (*fox* [f aa k s]), as *ck* (*jackal* [jh ae k el]) and as *cc* (*raccoon* [r ae k uw n]). Many other languages, for example, Spanish, are much more **transparent** in their sound-orthography mapping than English.

IPA

ARPAbet Symbol	IPA Symbol	Word	ARPAbet Transcription
[iy]	[i]	<u>lily</u>	[l ih l iy]
[ih]	[ɪ]	<u>lily</u>	[l ih l iy]
[ey]	[eɪ]	<u>daisy</u>	[d ey z iy]
[eh]	[ɛ]	<u>pen</u>	[p eh n]
[ae]	[æ]	<u>aster</u>	[ae s t axr]
[aa]	[ɑ]	<u>poppy</u>	[p aa p iy]
[ao]	[ɔ]	<u>orchid</u>	[ao r k ix d]
[uh]	[ʊ]	<u>wood</u>	[w uh d]
[ow]	[oʊ]	<u>lotus</u>	[l ow dx ax s]
[uw]	[u]	<u>tulip</u>	[t uw l ix p]
[ah]	[ʌ]	<u>buttercup</u>	[b ah dx axr k ah p]
[er]	[ɜː]	<u>bird</u>	[b er d]
[ay]	[aɪ]	<u>iris</u>	[ay r ix s]
[aw]	[aʊ]	<u>sunflower</u>	[s ah n f l aw axr]
[oy]	[oɪ]	<u>soil</u>	[s oy l]

Figure 27.2 ARPAbet symbols for American English vowels, with IPA equivalents.

27.2 Articulatory Phonetics

articulatory phonetics

Articulatory phonetics is the study of how these phones are produced as the various organs in the mouth, throat, and nose modify the airflow from the lungs.

glottis

voiced sound

unvoiced sound

nasal

consonant

vowel

27.2.1 The Vocal Organs

Figure 27.3 shows the organs of speech. Sound is produced by the rapid movement of air. Humans produce most sounds in spoken languages by expelling air from the lungs through the windpipe (technically, the **trachea**) and then out the mouth or nose. As it passes through the trachea, the air passes through the **larynx**, commonly known as the Adam's apple or voice box. The larynx contains two small folds of muscle, the **vocal folds** (often referred to non-technically as the **vocal cords**), which can be moved together or apart. The space between these two folds is called the **glottis**. If the folds are close together (but not tightly closed), they will vibrate as air passes through them; if they are far apart, they won't vibrate. Sounds made with the vocal folds together and vibrating are called **voiced**; sounds made without this vocal cord vibration are called **unvoiced** or **voiceless**. Voiced sounds include [b], [d], [g], [v], [z], and all the English vowels, among others. Unvoiced sounds include [p], [t], [k], [f], [s], and others.

The area above the trachea is called the **vocal tract**; it consists of the **oral tract** and the **nasal tract**. After the air leaves the trachea, it can exit the body through the mouth or the nose. Most sounds are made by air passing through the mouth. Sounds made by air passing through the nose are called **nasal sounds**; nasal sounds use both the oral and nasal tracts as resonating cavities; English nasal sounds include [m], [n], and [ng].

Phones are divided into two main classes: **consonants** and **vowels**. Both kinds of sounds are formed by the motion of air through the mouth, throat or nose. Consonants are made by restriction or blocking of the airflow in some way, and can be voiced or unvoiced. Vowels have less obstruction, are usually voiced, and are generally louder and longer-lasting than consonants. The technical use of these terms is



Figure 27.3 The vocal organs, shown in side view. Drawing by Laszlo Kubinyi from Sundberg (1977), ©Scientific American, used by permission.

much like the common usage; [p], [b], [t], [d], [k], [g], [f], [v], [s], [z], [r], [l], etc., are consonants; [aa], [ae], [ao], [ih], [aw], [ow], [uw], etc., are vowels. **Semivowels** (such as [y] and [w]) have some of the properties of both; they are voiced like vowels, but they are short and less syllabic like consonants.

27.2.2 Consonants: Place of Articulation

place of articulation

Because consonants are made by restricting the airflow in some way, consonants can be distinguished by where this restriction is made: the point of maximum restriction is called the **of a consonant**. Places of articulation, shown in Fig. 27.4, can be a useful way of grouping phones into equivalence classes, described below.

labial

Labial: Consonants whose main restriction is formed by the two lips coming together have a **bilabial** place of articulation. In English these include [p] as in *possum*, [b] as in *bear*, and [m] as in *marmot*. The English **labiodental**



Figure 27.4 Major English places of articulation.

consonants [v] and [f] are made by pressing the bottom lip against the upper row of teeth and letting the air flow through the space in the upper teeth.

dental **Dental:** Sounds that are made by placing the tongue against the teeth are dentals.

The main dentals in English are the [th] of *thing* and the [dh] of *thought*, which are made by placing the tongue behind the teeth with the tip slightly between the teeth.

alveolar **Alveolar:** The alveolar ridge is the portion of the roof of the mouth just behind the upper teeth. Most speakers of American English make the phones [s], [z], [t], and [d] by placing the tip of the tongue against the alveolar ridge. The word **coronal** is often used to refer to both dental and alveolar.

palatal **palate** **Palatal:** The roof of the mouth (the **palate**) rises sharply from the back of the alveolar ridge. The **palato-alveolar** sounds [sh] (*shrimp*), [ch] (*china*), [zh] (*Asian*), and [jh] (*jar*) are made with the blade of the tongue against the rising back of the alveolar ridge. The palatal sound [y] of *yak* is made by placing the front of the tongue up close to the palate.

velar **Velar:** The **velum**, or soft palate, is a movable muscular flap at the very back of the roof of the mouth. The sounds [k] (*cuckoo*), [g] (*goose*), and [ŋ] (*kingfisher*) are made by pressing the back of the tongue up against the velum.

glottal **Glottal:** The glottal stop [q] (IPA [ʔ]) is made by closing the glottis (by bringing the vocal folds together).

manner of articulation

27.2.3 Consonants: Manner of Articulation

Consonants are also distinguished by *how* the restriction in airflow is made, for example, by a complete stoppage of air or by a partial blockage. This feature is called the **manner of articulation** of a consonant. The combination of place and manner of articulation is usually sufficient to uniquely identify a consonant. Following are the major manners of articulation for English consonants:

stop

A **stop** is a consonant in which airflow is completely blocked for a short time. This blockage is followed by an explosive sound as the air is released. The period of blockage is called the **closure**, and the explosion is called the **release**. English has voiced stops like [b], [d], and [g] as well as unvoiced stops like [p], [t], and [k]. Stops are also called **plosives**.

nasal

The **nasal** sounds [n], [m], and [ŋ] are made by lowering the velum and allowing air to pass into the nasal cavity.

fricatives

In **fricatives**, airflow is constricted but not cut off completely. The turbulent airflow that results from the constriction produces a characteristic “hissing” sound. The English labiodental fricatives [f] and [v] are produced by pressing the lower lip against the upper teeth, allowing a restricted airflow between the upper teeth.

- sibilants** The dental fricatives [th] and [dh] allow air to flow around the tongue between the teeth. The alveolar fricatives [s] and [z] are produced with the tongue against the alveolar ridge, forcing air over the edge of the teeth. In the palato-alveolar fricatives [sh] and [zh], the tongue is at the back of the alveolar ridge, forcing air through a groove formed in the tongue. The higher-pitched fricatives (in English [s], [z], [sh] and [zh]) are called **sibilants**. Stops that are followed immediately by fricatives are called **africates**; these include English [ch] (*chicken*) and [jh] (*giraffe*).
- approximant** In **approximants**, the two articulators are close together but not close enough to cause turbulent airflow. In English [y] (*yellow*), the tongue moves close to the roof of the mouth but not close enough to cause the turbulence that would characterize a fricative. In English [w] (*wood*), the back of the tongue comes close to the velum. American [r] can be formed in at least two ways; with just the tip of the tongue extended and close to the palate or with the whole tongue bunched up near the palate. [l] is formed with the tip of the tongue up against the alveolar ridge or the teeth, with one or both sides of the tongue lowered to allow air to flow over it. [l] is called a **lateral** sound because of the drop in the sides of the tongue.
- tap** A **tap** or **flap** [dx] (or IPA [ɾ]) is a quick motion of the tongue against the alveolar ridge. The consonant in the middle of the word *lotus* ([l ɔ w dx ax s]) is a tap in most dialects of American English; speakers of many U.K. dialects would use a [t] instead of a tap in this word.

27.2.4 Vowels

Like consonants, vowels can be characterized by the position of the articulators as they are made. The three most relevant parameters for vowels are what is called vowel **height**, which correlates roughly with the height of the highest part of the tongue, vowel **frontness** or **backness**, indicating whether this high point is toward the front or back of the oral tract and whether the shape of the lips is **rounded** or not. Figure 27.5 shows the position of the tongue for different vowels.

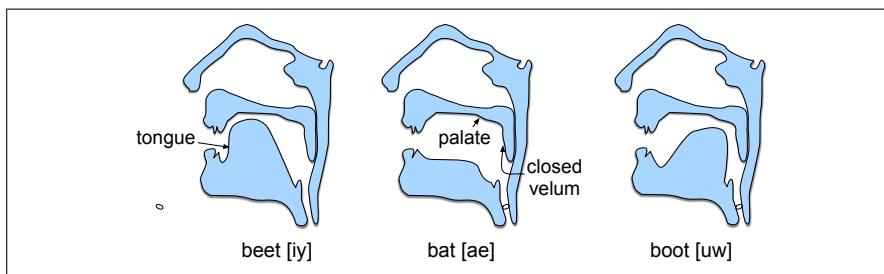


Figure 27.5 Positions of the tongue for three English vowels: high front [iy], low front [ae] and high back [uw].

- Front vowel** In the vowel [iy], for example, the highest point of the tongue is toward the front of the mouth. In the vowel [uw], by contrast, the high-point of the tongue is located toward the back of the mouth. Vowels in which the tongue is raised toward the front are called **front vowels**; those in which the tongue is raised toward the back are called **back vowels**. Note that while both [ih] and [eh] are front vowels, the tongue is higher for [ih] than for [eh]. Vowels in which the highest point of the tongue is comparatively high are called **high vowels**; vowels with mid or low values of maximum tongue height are called **mid vowels** or **low vowels**, respectively.
- Back vowel**
- High vowel**

Figure 27.6 shows a schematic characterization of the height of different vowels. It is schematic because the abstract property **height** correlates only roughly with ac-

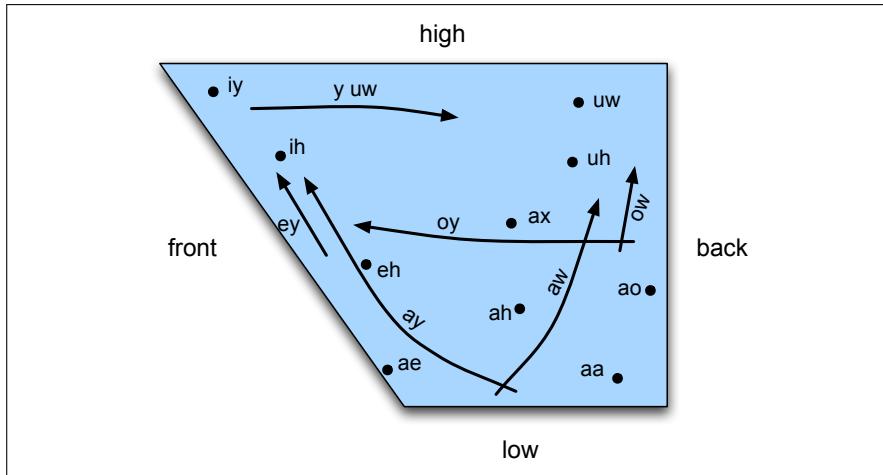


Figure 27.6 The schematic “vowel space” for English vowels.

tual tongue positions; it is, in fact, a more accurate reflection of acoustic facts. Note that the chart has two kinds of vowels: those in which tongue height is represented as a point and those in which it is represented as a path. A vowel in which the tongue position changes markedly during the production of the vowel is a **diphthong**. English is particularly rich in diphthongs.

The second important articulatory dimension for vowels is the shape of the lips. Certain vowels are pronounced with the lips rounded (the same lip shape used for whistling). These **rounded vowel**

diphthong

rounded vowel

27.2.5 Syllables

syllable

Consonants and vowels combine to make a **syllable**. A syllable is a vowel-like (or **sonorant**) sound together with some of the surrounding consonants that are most closely associated with it. The word *dog* has one syllable, [d aa g] (in our dialect); the word *catnip* has two syllables, [k ae t] and [n ih p]. We call the vowel at the core of a syllable the **nucleus**. The optional initial consonant or set of consonants is called the **onset**. If the onset has more than one consonant (as in the word *strike* [s t r a y k]), we say it has a **complex onset**. The **coda** is the optional consonant or sequence of consonants following the nucleus. Thus [d] is the onset of *dog*, and [g] is the coda. The **rime**, or **rhyme**, is the nucleus plus coda. Figure 27.7 shows some sample syllable structures.

nucleus

onset

coda

rime

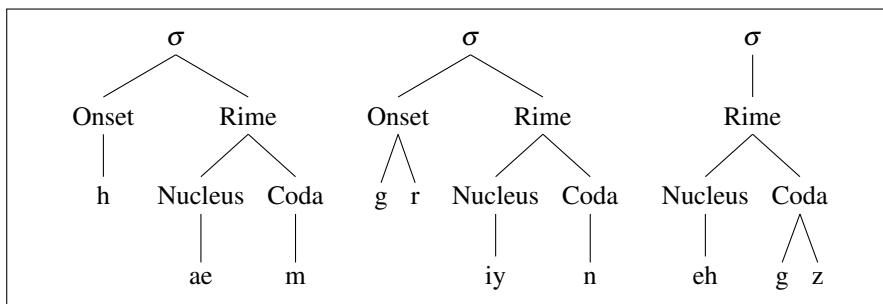


Figure 27.7 Syllable structure of *ham*, *green*, *eggs*. σ =syllable.

syllabification The task of automatically breaking up a word into syllables is called **syllabification**. Syllable structure is also closely related to the **phonotactics** of a language. The term **phonotactics** means the constraints on which phones can follow each other in a language. For example, English has strong constraints on what kinds of consonants can appear together in an onset; the sequence [zdr], for example, cannot be a legal English syllable onset. Phonotactics can be represented by a language model or finite-state model of phone sequences.

27.3 Prosodic Prominence: Accent, Stress and Schwa

prominence In a natural utterance of American English, some words sound more **prominent** than others, and certain syllables in these words are also more **prominent** than others. What we mean by prominence is that these words or syllables are perceptually more salient to the listener; speakers make a word or syllable more salient in English by saying it louder, saying it slower (so it has a longer duration), or by varying F0 during the word, making it higher or more variable.

pitch accent We capture the core notion of prominence by associating a linguistic marker with prominent words and syllables, a marker called **pitch accent**. Words or syllables that are prominent are said to **bear** (be associated with) a pitch accent. Pitch accent is thus part of the phonological description of a word in context in a spoken utterance.

Thus this utterance might be pronounced by **accenting** the underlined words:

(27.1) I'm a little surprised to hear it characterized as happy.

Nuclear Accent

emphatic accent We generally need more fine-grained distinctions than just a binary distinction between accented and unaccented words. For example, the last accent in a phrase generally is perceived as being more prominent than the other accents. This prominent last accent is called the **nuclear** or **emphatic accent**. Emphatic accents are generally used for semantic purposes, such as marking a word as the focus of the sentence or as contrastive or otherwise important in some way. Such emphatic words are often written IN CAPITAL LETTERS or with **stars** around them in texts or email or *Alice in Wonderland*; here's an example from the latter:

(27.2) "I know SOMETHING interesting is sure to happen," she said to herself.

Lexical Stress

lexical stress The syllables that bear pitch accent are called **accented** syllables, but not every syllable of a word can be accented. Pitch accent has to be realized on the syllable that has **lexical stress**. Lexical stress is a property of the words' pronunciation in dictionaries; the syllable that has lexical stress is the one that will be louder or longer if the word is accented. For example, the word *surprised* is stressed on its second syllable, not its first. (try stressing the other syllable by saying SURprised; hopefully that sounds wrong to you). Thus, if the word *surprised* receives a pitch accent in a sentence, it is the second syllable that will be stronger. The following example shows underlined accented words with the stressed syllable bearing the accent (the louder, longer syllable) in boldface:

(27.3) I'm a little surprised to hear it characterized as happy.

secondary stress

Stress can be marked in dictionaries in various ways. The CMU dictionary ([CMU, 1993](#)), for example, marks each vowel with the number 0 (unstressed), 1 (stressed), or 2 (secondary stress). Thus, the word *counter* is listed as [K AW1 N T ER0] and the word *table* as [T EY1 B AH0 L]. **secondary stress** is defined as a level of stress lower than primary stress but higher than an unstressed vowel, as in the word *dictionary* [D IH1 K SH AH0 N EH2 R IY0]. Difference in lexical stress can affect word meaning. For example the word *content* can be a noun or an adjective, but have different stressed syllables (the noun is pronounced [K AA1 N T EH0 N T] and the adjective [K AA0 N T EH1 N T]). In IPA, on the other hand, the symbol ['] before a syllable indicates that it has lexical stress (e.g., ['par.sli]).

reduced vowel schwa

Vowels that are unstressed can be weakened even further to **reduced vowels**. The most common reduced vowel is **schwa** ([ax]). Reduced vowels in English don't have their full form; the articulatory gesture isn't as complete as for a full vowel. As a result, the shape of the mouth is somewhat neutral; the tongue is neither particularly high nor low. The second vowel in *parakeet* is a schwa: [p ae r ax k iy t].

While schwa is the most common reduced vowel, it is not the only one, at least not in some dialects ([Bolinger, 1981](#)). Besides [ax], the ARPAbet also includes a reduced front vowel [ix] (IPA [i]), as well as [axr], which is an r-colored schwa (often called **schwar**).¹ Fig. 27.8 shows these reduced vowels.

ARPAbet Symbol	IPA Symbol	Word	ARPAbet Transcription
[ax]	[ə]	lotus	[l ow dx ax s]
[axr]	[əɹ]	heather	[h eh dh axr]
[ix]	[i]	tulip	[t uw l ix p]

Figure 27.8 Reduced vowels in American English, ARPAbet and IPA. [ax] is the reduced vowel schwa, [ix] is the reduced vowel corresponding to [ih], and [axr] is the reduced vowel corresponding to [er].

prominence

Not all unstressed vowels are reduced; any vowel, and diphthongs in particular, can retain its full quality even in unstressed position. For example, the vowel [iy] can appear in stressed position as in the word *eat* [iy t] or in unstressed position as in the word *carry* [k ae r iy].

We have mentioned a number of potential levels of **prominence**: accented, stressed, secondary stress, full vowel, and reduced vowel. It is still an open research question exactly how many levels are appropriate. Very few computational systems make use of all five of these levels, most using between one and three.

27.4 Prosodic Structure and Tune

prosody

In poetry, the word **prosody** refers to the study of the metrical structure of verse. In language processing, however, we use the term **prosody** to mean the study of the intonational and rhythmic aspects of language. More technically, prosody has been defined by [Ladd \(1996\)](#) as the “use of suprasegmental features to convey sentence-level pragmatic meanings”. The term **suprasegmental** means above and beyond the

suprasegmental

¹ [ix] is generally dropped in computational applications ([Miller, 1998](#)), and [ax] and [ix] are falling together in many dialects of English ([Wells, 1982, p. 167–168](#)).

level of the segment or phone. The term refers especially to the uses of acoustic features like **F0**, **duration**, and **energy** independently of the phone string.

By **sentence-level pragmatic meaning**, Ladd is referring to a number of kinds of meaning that have to do with the relation between a sentence and its discourse or external context. For example, prosody can be used to mark **discourse structure or function**, like the difference between statements and questions, or the way that a conversation is structured into segments or subdialogs. Prosody is also used to mark **saliency**, such as indicating that a particular word or phrase is important or salient. Finally, prosody is heavily used for affective and emotional meaning, such as expressing happiness, surprise, or anger.

The kind of prosodic prominence, that we saw in the prior section is one of the most computational studied aspects of prosody, but there are two others that we introduce in this section: **prosodic structure** and **tune**.

27.4.1 Prosodic Structure

prosodic phrasing
intonation phrase
intermediate phrase

tune

question rise
final fall

Spoken sentences have prosodic structure in the sense that some words seem to group naturally together and some words seem to have a noticeable break or disjunction between them. Prosodic structure is often described in terms of **prosodic phrasing**, meaning that an utterance has a prosodic phrase structure in a similar way to it having a syntactic phrase structure. For example, in the sentence *I wanted to go to London, but could only get tickets for France* there seem to be two main **intonation phrases**, their boundary occurring at the comma. Furthermore, in the first phrase, there seems to be another set of lesser prosodic phrase boundaries (often called **intermediate phrases**) that split up the words as *I wanted | to go | to London*.

There is also a correlation between prosodic structure and **syntactic structure** (Price et al. 1991, Ostendorf and Veilleux 1994, Koehn et al. 2000).

27.4.2 Tune

Two utterances with the same prominence and phrasing patterns can still differ prosodically by having different **tunes**. The **tune** of an utterance is the rise and fall of its F0 over time. A very obvious example of tune is the difference between statements and yes-no questions in English. The same sentence can be said with a final rise in F0 to indicate a yes-no question, or a final fall in F0 to indicate a declarative intonation. Figure 27.9 shows the F0 track of the same words spoken as a question or a statement. Note that the question rises at the end; this is often called a **question rise**. The falling intonation of the statement is called a **final fall**.

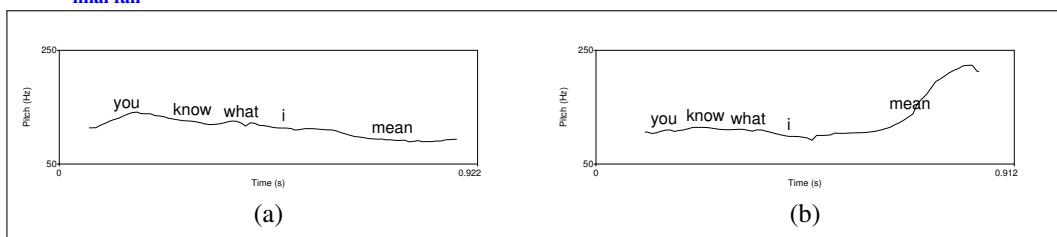


Figure 27.9 The same text read as the statement *You know what I mean* (on the left) and as a question *You know what I mean?* (on the right). Notice that yes-no question intonation in English has a sharp final rise in F0.

It turns out that English makes wide use of tune to express meaning. Besides this well-known rise for yes-no questions, an English phrase containing a list of nouns

continuation rise separated by commas often has a short rise called a **continuation rise** after each noun. Other examples include the characteristic English contours for expressing **contradiction** and expressing **surprise**.

The mapping between meaning and tune in English is extremely complex. Consider the utterance *oh, really*. Without varying the phrasing or stress, it is still possible to have many variants of this by varying the intonational tune. For example, we might have an excited version *oh, really!* (in the context of a reply to a statement that you've just won the lottery); a sceptical version *oh, really?*—in the context of not being sure that the speaker is being honest; to an angry *oh, really!* indicating displeasure.

Linking Tune with Prominence: ToBI

It is also possible to link models of prominence with models of tune, allowing us to model differences between pitch accents according to the **tune** associated with them.

One of the most widely used linguistic models of prosody that enables this association is the **ToBI** (Tone and Break Indices) model (Silverman et al. 1992, Beckman and Hirschberg 1994, Pierrehumbert 1980, Pitrelli et al. 1994). ToBI is a phonological theory of intonation that models prominence, tune, and boundaries. ToBI's model of prominence and tunes is based on the five **pitch accents** and four **boundary tones** shown in Fig. 27.10.

Pitch Accents		Boundary Tones	
H*	peak accent	L-L%	“final fall”: “declarative contour” of American English
L*	low accent	L-H%	continuation rise
L*+H	scooped accent	H-H%	“question rise”: canonical yes-no question contour
L+H*	rising peak accent	H-L%	final level plateau (plateau because H- causes “upstep” of following)
H+!H*	step down		

Figure 27.10 The accent and boundary tones labels from the ToBI transcription system for American English intonation (Beckman and Ayers 1997, Beckman and Hirschberg 1994).

An utterance in ToBI consists of a sequence of intonational phrases, each of which ends in one of the four **boundary tones**. The boundary tones represent the utterance final aspects of tune. Each word in the utterances can optionally be associated with one of the five types of pitch accents.

Each intonational phrase consists of one or more **intermediate phrase**. These phrases can also be marked with kinds of boundary tone, including the **%H** high initial boundary tone, which marks a phrase that is particularly high in the speaker's pitch range, as well as final phrase accents **H-** and **L-**.

In addition to accents and boundary tones, ToBI distinguishes four levels of phrasing, labeled on a separate **break index** tier. The largest phrasal breaks are the intonational phrase (break index **4**) and the intermediate phrase (break index **3**), discussed above. Break index **2** is used to mark a disjunction or pause between words that is smaller than an intermediate phrase, and **1** is used for normal phrase-medial word boundaries.

Figure 27.11 shows the tone, orthographic, and phrasing **tiers** of a ToBI transcription, using the Praat program. The same sentence is read with two different tunes. In (a), the word *Marianna* is spoken with a high **H*** accent, and the sentence has the declarative boundary tone **L-L%**. In (b), the word *Marianna* is spoken with a

low L* accent and the yes-no question boundary tone H-H%. One goal of ToBI is to express different meanings to the different type of accents. Here, the L* accent adds a meaning of *surprise* to the sentence (i.e., with a connotation like ‘Are you really saying it was Marianna?’) (Hirschberg and Pierrehumbert 1986, Steedman 2007).



Figure 27.11 The same sentence read by Mary Beckman with two different intonation patterns and transcribed in ToBI. (a) Shows an H* accent and the typical American English declarative final fall L-L%. (b) Shows the L* accent, with the typical American English yes-no question rise H-H%.

ToBI models have been proposed for many languages (Jun, 2005), such as the J_TOBI system for Japanese (Venditti, 2005).

27.5 Acoustic Phonetics and Signals

We begin with a brief introduction to the acoustic waveform and how it is digitized and summarize the idea of frequency analysis and spectra. This is an extremely brief overview; the interested reader is encouraged to consult the references at the end of the chapter.

27.5.1 Waves

Acoustic analysis is based on the sine and cosine functions. Figure 27.12 shows a plot of a sine wave, in particular the function

$$y = A * \sin(2\pi ft) \quad (27.4)$$

where we have set the amplitude A to 1 and the frequency f to 10 cycles per second.

Recall from basic mathematics that two important characteristics of a wave are its **frequency** and **amplitude**. The frequency is the number of times a second that a wave repeats itself, that is, the number of **cycles**. We usually measure frequency in **cycles per second**. The signal in Fig. 27.12 repeats itself 5 times in .5 seconds, hence 10 cycles per second. Cycles per second are usually called **hertz** (shortened to **Hz**), so the frequency in Fig. 27.12 would be described as 10 Hz. The **amplitude** A of a sine wave is the maximum value on the Y axis.

The **period** T of the wave is defined as the time it takes for one cycle to complete, defined as

$$T = \frac{1}{f} \quad (27.5)$$



Figure 27.12 A sine wave with a frequency of 10 Hz and an amplitude of 1.

In Fig. 27.12 we can see that each cycle lasts a tenth of a second; hence $T = .1$ seconds.

27.5.2 Speech Sound Waves

Let's turn from hypothetical waves to sound waves. The input to a speech recognizer, like the input to the human ear, is a complex series of changes in air pressure. These changes in air pressure obviously originate with the speaker and are caused by the specific way that air passes through the glottis and out the oral or nasal cavities. We represent sound waves by plotting the change in air pressure over time. One metaphor which sometimes helps in understanding these graphs is that of a vertical plate blocking the air pressure waves (perhaps in a microphone in front of a speaker's mouth, or the eardrum in a hearer's ear). The graph measures the amount of **compression** or **rarefaction** (uncompression) of the air molecules at this plate. Figure 27.13 shows a short segment of a waveform taken from the Switchboard corpus of telephone speech of the vowel [iy] from someone saying “she just had a baby”.

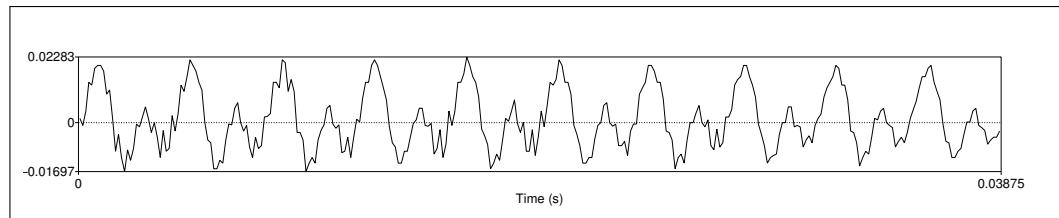


Figure 27.13 A waveform of the vowel [iy] from an utterance shown later in Fig. 27.17 on page 534. The y-axis shows the level of air pressure above and below normal atmospheric pressure. The x-axis shows time. Notice that the wave repeats regularly.

sampling

Let's explore how the digital representation of the sound wave shown in Fig. 27.13 would be constructed. The first step in processing speech is to convert the analog representations (first air pressure and then analog electric signals in a microphone) into a digital signal. This process of **analog-to-digital conversion** has two steps: **sampling** and **quantization**. To sample a signal, we measure its amplitude at a particular time; the **sampling rate** is the number of samples taken per second. To accurately measure a wave, we must have at least two samples in each cycle: one measuring the positive part of the wave and one measuring the negative part. More than two samples per cycle increases the amplitude accuracy, but fewer than two samples causes the frequency of the wave to be completely missed. Thus, the maxi-

Nyquist frequency

mum frequency wave that can be measured is one whose frequency is half the sample rate (since every cycle needs two samples). This maximum frequency for a given sampling rate is called the **Nyquist frequency**. Most information in human speech is in frequencies below 10,000 Hz; thus, a 20,000 Hz sampling rate would be necessary for complete accuracy. But telephone speech is filtered by the switching network, and only frequencies less than 4,000 Hz are transmitted by telephones. Thus, an 8,000 Hz sampling rate is sufficient for **telephone-bandwidth** speech like the Switchboard corpus. A 16,000 Hz sampling rate (sometimes called **wideband**) is often used for microphone speech.

quantization

Even an 8,000 Hz sampling rate requires 8000 amplitude measurements for each second of speech, so it is important to store amplitude measurements efficiently. They are usually stored as integers, either 8 bit (values from -128–127) or 16 bit (values from -32768–32767). This process of representing real-valued numbers as integers is called **quantization** because the difference between two integers acts as a minimum granularity (a quantum size) and all values that are closer together than this quantum size are represented identically.

channel

Once data is quantized, it is stored in various formats. One parameter of these formats is the sample rate and sample size discussed above; telephone speech is often sampled at 8 kHz and stored as 8-bit samples, and microphone data is often sampled at 16 kHz and stored as 16-bit samples. Another parameter of these formats is the number of **channels**. For stereo data or for two-party conversations, we can store both channels in the same file or we can store them in separate files. A final parameter is individual sample storage—linearly or compressed. One common compression format used for telephone speech is μ -law (often written u-law but still pronounced mu-law). The intuition of log compression algorithms like μ -law is that human hearing is more sensitive at small intensities than large ones; the log represents small values with more faithfulness at the expense of more error on large values. The linear (unlogged) values are generally referred to as **linear PCM** values (PCM stands for pulse code modulation, but never mind that). Here's the equation for compressing a linear PCM sample value x to 8-bit μ -law, (where $\mu=255$ for 8 bits):

PCM

$$F(x) = \frac{\operatorname{sgn}(s) \log(1 + \mu|s|)}{\log(1 + \mu)} \quad (27.6)$$

There are a number of standard file formats for storing the resulting digitized waveform, such as Microsoft's .wav, Apple's AIFF and Sun's AU, all of which have special headers; simple headerless “raw” files are also used. For example, the .wav format is a subset of Microsoft's RIFF format for multimedia files; RIFF is a general format that can represent a series of nested chunks of data and control information. Figure 27.14 shows a simple .wav file with a single data chunk together with its format chunk.



Figure 27.14 Microsoft wavefile header format, assuming simple file with one chunk. Following this 44-byte header would be the data chunk.

27.5.3 Frequency and Amplitude; Pitch and Loudness

Sound waves, like all waves, can be described in terms of frequency, amplitude, and the other characteristics that we introduced earlier for pure sine waves. In sound waves, these are not quite as simple to measure as they were for sine waves. Let's consider frequency. Note in Fig. 27.13 that although not exactly a sine, the wave is nonetheless periodic, repeating 10 times in the 38.75 milliseconds (.03875 seconds) captured in the figure. Thus, the frequency of this segment of the wave is $10/.03875$ or 258 Hz.

Where does this periodic 258 Hz wave come from? It comes from the speed of vibration of the vocal folds; since the waveform in Fig. 27.13 is from the vowel [iy], it is voiced. Recall that voicing is caused by regular openings and closing of the vocal folds. When the vocal folds are open, air is pushing up through the lungs, creating a region of high pressure. When the folds are closed, there is no pressure from the lungs. Thus, when the vocal folds are vibrating, we expect to see regular peaks in amplitude of the kind we see in Fig. 27.13, each major peak corresponding to an opening of the vocal folds. The frequency of the vocal fold vibration, or the frequency of the complex wave, is called the **fundamental frequency** of the waveform, often abbreviated **F0**. We can plot F0 over time in a **pitch track**. Figure 27.15 shows the pitch track of a short question, “Three o’clock?” represented below the waveform. Note the rise in F0 at the end of the question.

fundamental frequency
F0
pitch track

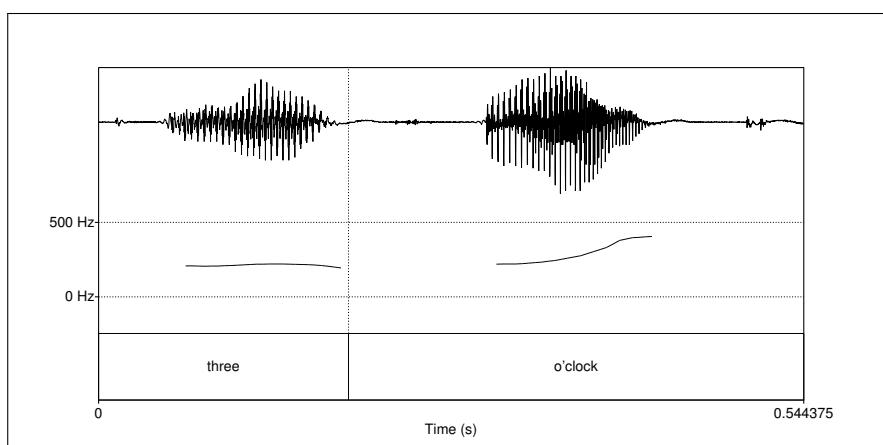


Figure 27.15 Pitch track of the question “Three o’clock?”, shown below the wavefile. Note the rise in F0 at the end of the question. Note the lack of pitch trace during the very quiet part (the “o” of “o’clock”; automatic pitch tracking is based on counting the pulses in the voiced regions, and doesn’t work if there is no voicing (or insufficient sound).

The vertical axis in Fig. 27.13 measures the amount of air pressure variation; pressure is force per unit area, measured in Pascals (Pa). A high value on the vertical axis (a high amplitude) indicates that there is more air pressure at that point in time, a zero value means there is normal (atmospheric) air pressure, and a negative value means there is lower than normal air pressure (rarefaction).

In addition to this value of the amplitude at any point in time, we also often need to know the average amplitude over some time range, to give us some idea of how great the average displacement of air pressure is. But we can’t just take the average of the amplitude values over a range; the positive and negative values would (mostly) cancel out, leaving us with a number close to zero. Instead, we generally use the RMS (root-mean-square) amplitude, which squares each number

before averaging (making it positive), and then takes the square root at the end.

$$\text{RMS amplitude}_{i=1}^N = \sqrt{\frac{1}{N} \sum_{i=1}^N x_i^2} \quad (27.7)$$

Power

The **power** of the signal is related to the square of the amplitude. If the number of samples of a sound is N , the power is

$$\text{Power} = \frac{1}{N} \sum_{i=1}^N x_i^2 \quad (27.8)$$

Intensity

Rather than power, we more often refer to the **intensity** of the sound, which normalizes the power to the human auditory threshold and is measured in dB. If P_0 is the auditory threshold pressure = 2×10^{-5} Pa, then intensity is defined as follows:

$$\text{Intensity} = 10 \log_{10} \frac{1}{NP_0} \sum_{i=1}^N x_i^2 \quad (27.9)$$

Figure 27.16 shows an intensity plot for the sentence “Is it a long movie?” from the CallHome corpus, again shown below the waveform plot.

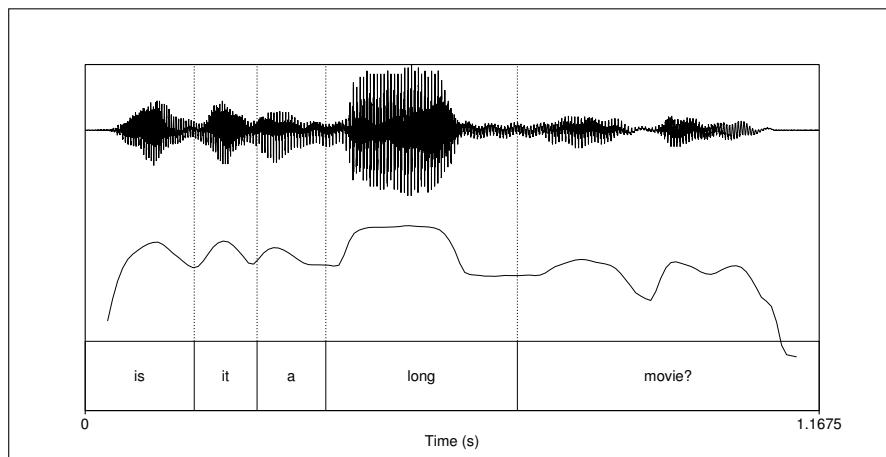


Figure 27.16 Intensity plot for the sentence “Is it a long movie?”. Note the intensity peaks at each vowel and the especially high peak for the word *long*.

pitch

Two important perceptual properties, **pitch** and **loudness**, are related to frequency and intensity. The **pitch** of a sound is the mental sensation, or perceptual correlate, of fundamental frequency; in general, if a sound has a higher fundamental frequency we perceive it as having a higher pitch. We say “in general” because the relationship is not linear, since human hearing has different acuities for different frequencies. Roughly speaking, human pitch perception is most accurate between 100 Hz and 1000 Hz and in this range pitch correlates linearly with frequency. Human hearing represents frequencies above 1000 Hz less accurately, and above this range, pitch correlates logarithmically with frequency. Logarithmic representation means that the differences between high frequencies are compressed and hence not as accurately perceived. There are various psychoacoustic models of pitch perception scales. One common model is the **mel** scale (Stevens et al. 1937, Stevens and

Mel

[Volkmann 1940](#)). A mel is a unit of pitch defined such that pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of mels. The mel frequency m can be computed from the raw acoustic frequency as follows:

$$m = 1127 \ln\left(1 + \frac{f}{700}\right) \quad (27.10)$$

As we'll see in Chapter 28, the mel scale plays an important role in speech recognition.

The **loudness** of a sound is the perceptual correlate of the **power**. So sounds with higher amplitudes are perceived as louder, but again the relationship is not linear. First of all, as we mentioned above when we defined μ -law compression, humans have greater resolution in the low-power range; the ear is more sensitive to small power differences. Second, it turns out that there is a complex relationship between power, frequency, and perceived loudness; sounds in certain frequency ranges are perceived as being louder than those in other frequency ranges.

pitch extraction Various algorithms exist for automatically extracting F0. In a slight abuse of terminology, these are called **pitch extraction** algorithms. The autocorrelation method of pitch extraction, for example, correlates the signal with itself at various offsets. The offset that gives the highest correlation gives the period of the signal. Other methods for pitch extraction are based on the cepstral features we introduce in Chapter 28. There are various publicly available pitch extraction toolkits; for example, an augmented autocorrelation pitch tracker is provided with Praat ([Boersma and Weenink, 2005](#)).

27.5.4 Interpretation of Phones from a Waveform

Much can be learned from a visual inspection of a waveform. For example, vowels are pretty easy to spot. Recall that vowels are voiced; another property of vowels is that they tend to be long and are relatively loud (as we can see in the intensity plot in Fig. 27.16). Length in time manifests itself directly on the x-axis, and loudness is related to (the square of) amplitude on the y-axis. We saw in the previous section that voicing is realized by regular peaks in amplitude of the kind we saw in Fig. 27.13, each major peak corresponding to an opening of the vocal folds. Figure 27.17 shows the waveform of the short sentence “she just had a baby”. We have labeled this waveform with word and phone labels. Notice that each of the six vowels in Fig. 27.17, [iy], [ax], [ae], [ax], [ey], [iy], all have regular amplitude peaks indicating voicing.



Figure 27.17 A waveform of the sentence “She just had a baby” from the Switchboard corpus (conversation 4325). The speaker is female, was 20 years old in 1991, which is approximately when the recording was made, and speaks the South Midlands dialect of American English.

For a stop consonant, which consists of a closure followed by a release, we can often see a period of silence or near silence followed by a slight burst of amplitude. We can see this for both of the [b]’s in *baby* in Fig. 27.17.

Another phone that is often quite recognizable in a waveform is a fricative. Recall that fricatives, especially very strident fricatives like [sh], are made when a narrow channel for airflow causes noisy, turbulent air. The resulting hissy sounds have a noisy, irregular waveform. This can be seen somewhat in Fig. 27.17; it’s even clearer in Fig. 27.18, where we’ve magnified just the first word *she*.



Figure 27.18 A more detailed view of the first word “she” extracted from the wavefile in Fig. 27.17. Notice the difference between the random noise of the fricative [sh] and the regular voicing of the vowel [iy].

27.5.5 Spectra and the Frequency Domain

While some broad phonetic features (such as energy, pitch, and the presence of voicing, stop closures, or fricatives) can be interpreted directly from the waveform, most computational applications such as speech recognition (as well as human auditory processing) are based on a different representation of the sound in terms of its component frequencies. The insight of **Fourier analysis** is that every complex wave can be represented as a sum of many sine waves of different frequencies. Consider the waveform in Fig. 27.19. This waveform was created (in Praat) by summing two sine waveforms, one of frequency 10 Hz and one of frequency 100 Hz.

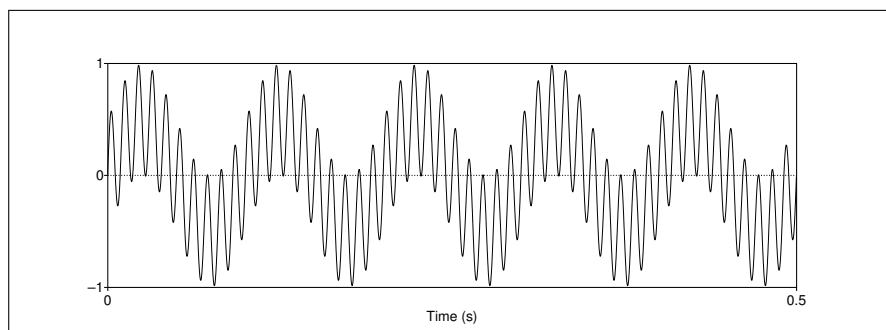


Figure 27.19 A waveform that is the sum of two sine waveforms, one of frequency 10 Hz (note five repetitions in the half-second window) and one of frequency 100 Hz, both of amplitude 1.

spectrum

We can represent these two component frequencies with a **spectrum**. The spectrum of a signal is a representation of each of its frequency components and their amplitudes. Figure 27.20 shows the spectrum of Fig. 27.19. Frequency in Hz is on the x-axis and amplitude on the y-axis. Note the two spikes in the figure, one

at 10 Hz and one at 100 Hz. Thus, the spectrum is an alternative representation of the original waveform, and we use the spectrum as a tool to study the component frequencies of a sound wave at a particular time point.

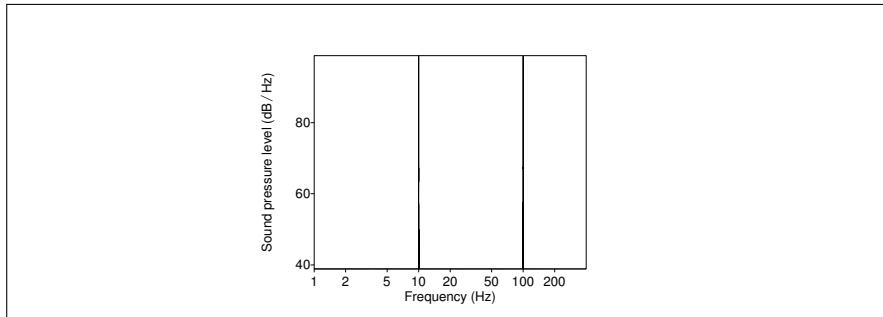


Figure 27.20 The spectrum of the waveform in Fig. 27.19.

Let's look now at the frequency components of a speech waveform. Figure 27.21 shows part of the waveform for the vowel [ae] of the word *had*, cut out from the sentence shown in Fig. 27.17.

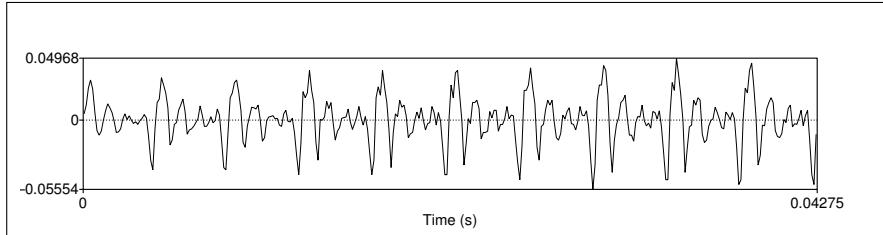


Figure 27.21 The waveform of part of the vowel [ae] from the word *had* cut out from the waveform shown in Fig. 27.17.

Note that there is a complex wave that repeats about ten times in the figure; but there is also a smaller repeated wave that repeats four times for every larger pattern (notice the four small peaks inside each repeated wave). The complex wave has a frequency of about 234 Hz (we can figure this out since it repeats roughly 10 times in .0427 seconds, and 10 cycles/.0427 seconds = 234 Hz).

The smaller wave then should have a frequency of roughly four times the frequency of the larger wave, or roughly 936 Hz. Then, if you look carefully, you can see two little waves on the peak of many of the 936 Hz waves. The frequency of this tiniest wave must be roughly twice that of the 936 Hz wave, hence 1872 Hz.

Figure 27.22 shows a smoothed spectrum for the waveform in Fig. 27.21, computed with a discrete Fourier transform (DFT).

The *x*-axis of a spectrum shows frequency, and the *y*-axis shows some measure of the magnitude of each frequency component (in decibels (dB), a logarithmic measure of amplitude that we saw earlier). Thus, Fig. 27.22 shows significant frequency components at around 930 Hz, 1860 Hz, and 3020 Hz, along with many other lower-magnitude frequency components. These first two components are just what we noticed in the time domain by looking at the wave in Fig. 27.21!

Why is a spectrum useful? It turns out that these spectral peaks that are easily visible in a spectrum are characteristic of different phones; phones have characteristic spectral “signatures”. Just as chemical elements give off different wavelengths of light when they burn, allowing us to detect elements in stars by looking at the spec-



Figure 27.22 A spectrum for the vowel [ae] from the word *had* in the waveform of *She just had a baby* in Fig. 27.17.

trum of the light, we can detect the characteristic signature of the different phones by looking at the spectrum of a waveform. This use of spectral information is essential to both human and machine speech recognition. In human audition, the function of the **cochlea**, or **inner ear**, is to compute a spectrum of the incoming waveform. Similarly, the various kinds of acoustic features used in speech recognition as the HMM observation are all different representations of spectral information.

Let's look at the spectrum of different vowels. Since some vowels change over time, we'll use a different kind of plot called a **spectrogram**. While a spectrum shows the frequency components of a wave at one point in time, a **spectrogram** is a way of envisioning how the different frequencies that make up a waveform change over time. The *x*-axis shows time, as it did for the waveform, but the *y*-axis now shows frequencies in hertz. The darkness of a point on a spectrogram corresponds to the amplitude of the frequency component. Very dark points have high amplitude, light points have low amplitude. Thus, the spectrogram is a useful way of visualizing the three dimensions (time \times frequency \times amplitude).

Figure 27.23 shows spectrograms of three American English vowels, [ih], [ae], and [ah]. Note that each vowel has a set of dark bars at various frequency bands, slightly different bands for each vowel. Each of these represents the same kind of spectral peak that we saw in Fig. 27.21.

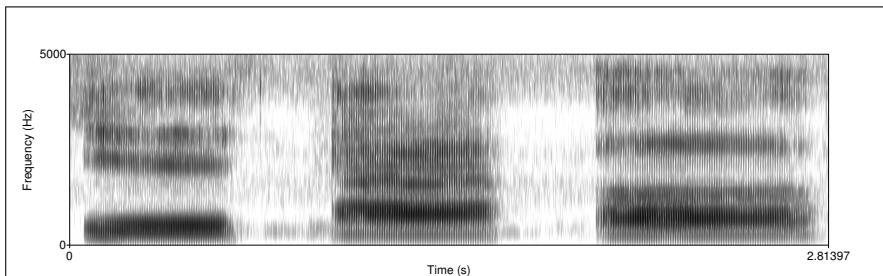


Figure 27.23 Spectrograms for three American English vowels, [ih], [ae], and [uh], spoken by the first author.

Each dark bar (or spectral peak) is called a **formant**. As we discuss below, a formant is a frequency band that is particularly amplified by the vocal tract. Since different vowels are produced with the vocal tract in different positions, they will produce different kinds of amplifications or resonances. Let's look at the first two formants, called F1 and F2. Note that F1, the dark bar closest to the bottom, is in a different position for the three vowels; it's low for [ih] (centered at about 470 Hz)

and somewhat higher for [ae] and [ah] (somewhere around 800 Hz). By contrast, F2, the second dark bar from the bottom, is highest for [ih], in the middle for [ae], and lowest for [ah].

We can see the same formants in running speech, although the reduction and coarticulation processes make them somewhat harder to see. Figure 27.24 shows the spectrogram of “she just had a baby”, whose waveform was shown in Fig. 27.17. F1 and F2 (and also F3) are pretty clear for the [ax] of *just*, the [ae] of *had*, and the [ey] of *baby*.

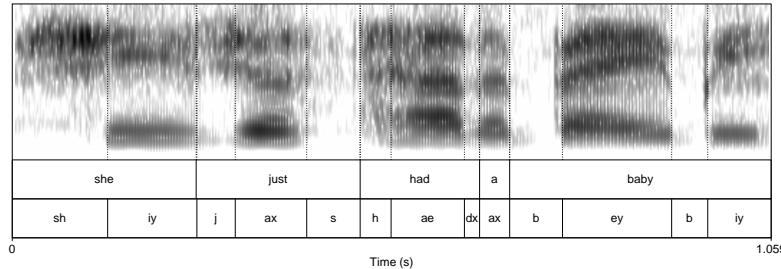


Figure 27.24 A spectrogram of the sentence “she just had a baby” whose waveform was shown in Fig. 27.17. We can think of a spectrogram as a collection of spectra (time slices), like Fig. 27.22 placed end to end.

What specific clues can spectral representations give for phone identification? First, since different vowels have their formants at characteristic places, the spectrum can distinguish vowels from each other. We’ve seen that [ae] in the sample waveform had formants at 930 Hz, 1860 Hz, and 3020 Hz. Consider the vowel [iy] at the beginning of the utterance in Fig. 27.17. The spectrum for this vowel is shown in Fig. 27.25. The first formant of [iy] is 540 Hz, much lower than the first formant for [ae], and the second formant (2581 Hz) is much higher than the second formant for [ae]. If you look carefully, you can see these formants as dark bars in Fig. 27.24 just around 0.5 seconds.



Figure 27.25 A smoothed (LPC) spectrum for the vowel [iy] at the start of *She just had a baby*. Note that the first formant (540 Hz) is much lower than the first formant for [ae] shown in Fig. 27.22, and the second formant (2581 Hz) is much higher than the second formant for [ae].

The location of the first two formants (called F1 and F2) plays a large role in determining vowel identity, although the formants still differ from speaker to speaker.

Higher formants tend to be caused more by general characteristics of a speaker's vocal tract rather than by individual vowels. Formants also can be used to identify the nasal phones [n], [m], and [ng] and the liquids [l] and [r].

27.5.6 The Source-Filter Model

source-filter model

harmonic

Why do different vowels have different spectral signatures? As we briefly mentioned above, the formants are caused by the resonant cavities of the mouth. The **source-filter model** is a way of explaining the acoustics of a sound by modeling how the pulses produced by the glottis (the **source**) are shaped by the vocal tract (the **filter**).

Let's see how this works. Whenever we have a wave such as the vibration in air caused by the glottal pulse, the wave also has **harmonics**. A harmonic is another wave whose frequency is a multiple of the fundamental wave. Thus, for example, a 115 Hz glottal fold vibration leads to harmonics (other waves) of 230 Hz, 345 Hz, 460 Hz, and so on on. In general, each of these waves will be weaker, that is, will have much less amplitude than the wave at the fundamental frequency.

It turns out, however, that the vocal tract acts as a kind of filter or amplifier; indeed any cavity, such as a tube, causes waves of certain frequencies to be amplified and others to be damped. This amplification process is caused by the shape of the cavity; a given shape will cause sounds of a certain frequency to resonate and hence be amplified. Thus, by changing the shape of the cavity, we can cause different frequencies to be amplified.

When we produce particular vowels, we are essentially changing the shape of the vocal tract cavity by placing the tongue and the other articulators in particular positions. The result is that different vowels cause different harmonics to be amplified. So a wave of the same fundamental frequency passed through different vocal tract positions will result in different harmonics being amplified.

We can see the result of this amplification by looking at the relationship between the shape of the vocal tract and the corresponding spectrum. Figure 27.26 shows the vocal tract position for three vowels and a typical resulting spectrum. The formants are places in the spectrum where the vocal tract happens to amplify particular harmonic frequencies.

27.6 Phonetic Resources

Pronunciation dictionary

A wide variety of phonetic resources can be drawn on for computational work. One key set of resources are **pronunciation dictionaries**. Such on-line phonetic dictionaries give phonetic transcriptions for each word. Three commonly used on-line dictionaries for English are the CELEX, CMUDict, and PRONLEX lexicons; for other languages, the LDC has released pronunciation dictionaries for Egyptian Arabic, German, Japanese, Korean, Mandarin, and Spanish. All these dictionaries can be used for both speech recognition and synthesis work.

The CELEX dictionary ([Baayen et al., 1995](#)) is the most richly annotated of the dictionaries. It includes all the words in the 1974 Oxford Advanced Learner's Dictionary (41,000 lemmata) and the 1978 Longman Dictionary of Contemporary English (53,000 lemmata); in total it has pronunciations for 160,595 wordforms. Its (British rather than American) pronunciations are transcribed with an ASCII version of the IPA called SAM. In addition to basic phonetic information like phone strings, syllabification, and stress level for each syllable, each word is also annotated with



Figure 27.26 Visualizing the vocal tract position as a filter: the tongue positions for three English vowels and the resulting smoothed spectra showing F1 and F2.

morphological, part-of-speech, syntactic, and frequency information. CELEX (as well as CMU and PRONLEX) represent three levels of stress: primary stress, secondary stress, and no stress. For example, some of the CELEX information for the word *dictionary* includes multiple pronunciations ('dIk-S@n-rI and 'dIk-S@-n@-rI, corresponding to ARPAbet [d ih k sh ax n r ih] and [d ih k sh ax n ax r ih], respectively), together with the CV skelata for each one ([CVC][CVC][CV] and [CVC][CV][CV][CV]), the frequency of the word, the fact that it is a noun, and its morphological structure (diction+ary).

The free CMU Pronouncing Dictionary (CMU, 1993) has pronunciations for about 125,000 wordforms. It uses a 39-phone ARPAbet-derived phoneme set. Transcriptions are phonemic, and thus instead of marking any kind of surface reduction like flapping or reduced vowels, it marks each vowel with the number 0 (unstressed), 1 (stressed), or 2 (secondary stress). Thus, the word *tiger* is listed as [T AY1 G ER0], the word *table* as [T EY1 B AH0 L], and the word *dictionary* as [D IH1 K SH AH0 N EH2 R IY0]. The dictionary is not syllabified, although the nucleus is implicitly marked by the (numbered) vowel. Figure 27.27 shows some sample pronunciations.

ANTECEDENTS	AE2 N T IH0 S IY1 D AH0 N T S	PAKISTANI	P AE2 K IH0 S T AE1 N IY0
CHANG	CH AE1 NG	TABLE	T EY1 B AH0 L
DICTIONARY	D IH1 K SH AH0 N EH2 R IY0	TROTSKY	T R AA1 T S K IY2
DINNER	D IH1 N ER0	WALTER	W AO1 L T ER0
LUNCH	L AH1 N CH	WALTZING	W AO1 L T S IH0 NG
MCFARLAND	M AH0 K F AA1 R L AH0 N D	WALTZING(2)	W AO1 L S IH0 NG

Figure 27.27 Some sample pronunciations from the CMU Pronouncing Dictionary.

The PRONLEX dictionary ([LDC, 1995](#)) was designed for speech recognition and contains pronunciations for 90,694 wordforms. It covers all the words used in many years of the *Wall Street Journal*, as well as the Switchboard Corpus. PRONLEX has the advantage that it includes many proper names (20,000, whereas CELEX only has about 1000). Names are important for practical applications, and they are both frequent and difficult; we return to a discussion of deriving name pronunciations in Chapter 28.

The CMU dictionary was designed for speech recognition rather than synthesis uses; thus, it does not specify which of the multiple pronunciations to use for synthesis, does not mark syllable boundaries, and because it capitalizes the dictionary headwords, does not distinguish between, for example, *US* and *us* (the form *US* has the two pronunciations [AH1 S] and [Y UW1 EH1 S]).

The 110,000 word UNISYN dictionary, freely available for research purposes, resolves many of these issues as it was designed specifically for synthesis ([Fitt, 2002](#)). UNISYN gives syllabifications, stress, and some morphological boundaries. Furthermore, pronunciations in UNISYN can also be read off in any of dozens of dialects of English, including General American, RP British, Australia, and so on. The UNISYN uses a slightly different phone set; here are some examples:

```
going: { g * ou }.> i ng >
antecedents: { * a n . t^ i . s ~ ii . d n! t }> s >
dictionary: { d * i k . sh @ . n ~ e . r ii }
```

Another useful resource is a **phonetically annotated corpus**, in which a collection of waveforms is hand-labeled with the corresponding string of phones. Three important phonetic corpora in English are the TIMIT corpus, the Switchboard corpus, and the Buckeye corpus.

The TIMIT corpus ([NIST, 1990](#)) was collected as a joint project between Texas Instruments (TI), MIT, and SRI. It is a corpus of 6300 read sentences, with 10 sentences each from 630 speakers. The 6300 sentences were drawn from a set of 2342 predesigned sentences, some selected to have particular dialect shibboleths, others to maximize phonetic diphone coverage. Each sentence in the corpus was phonetically hand-labeled, the sequence of phones was automatically aligned with the sentence waveform, and then the automatic phone boundaries were manually hand-corrected ([Seneff and Zue, 1988](#)). The result is a **time-aligned transcription**: a transcription in which each phone is associated with a start and end time in the waveform. We showed a graphical example of a time-aligned transcription in Fig. 27.17 on page 534.

time-aligned transcription

The phoneset for TIMIT and for the Switchboard Transcription Project corpus below, is a more detailed one than the minimal phonemic version of the ARPAbet. In particular, these phonetic transcriptions make use of the various reduced and rare phones mentioned in Fig. 27.1 and Fig. 27.2: the flap [dx], glottal stop [q], reduced vowels [ax], [ix], [axr], voiced allophone of [h] ([hv]), and separate phones for stop closure ([dcl], [tcl], etc) and release ([d], [t], etc.). An example transcription is shown in Fig. 27.28.

she	had	your	dark	suit	in	greasy	wash	water	all	year
sh iy	hv ae dcl	jh axr	dcl d aa r kcl	s ux q	en	gcl g r iy s ix	w aa sh	q w aa dx axr q	aa l	y ix axr

Figure 27.28 Phonetic transcription from the TIMIT corpus. This transcription uses special features of ARPAbet for narrow transcription, such as the palatalization of [d] in *had*, unreleased final stop in *dark*, glottalization of final [t] in *suit* to [q], and flap of [t] in *water*. The TIMIT corpus also includes time-alignments for each phone (not shown).

Where TIMIT is based on read speech, the more recent Switchboard Transcription Project corpus is based on the Switchboard corpus of conversational speech. This phonetically annotated portion consists of approximately 3.5 hours of sentences extracted from various conversations (Greenberg et al., 1996). As with TIMIT, each annotated utterance contains a time-aligned transcription. The Switchboard transcripts are time aligned at the syllable level rather than at the phone level; thus, a transcript consists of a sequence of syllables with the start and end time of each syllables in the corresponding wavefile. Figure 27.29 shows an example from the Switchboard Transcription Project for the phrase *they're kind of in between right now*.

0.470	0.640	0.720	0.900	0.953	1.279	1.410	1.630
dh er	k aa	n ax	v ih m	b ix	t w iy n	r ay	n aw

Figure 27.29 Phonetic transcription of the Switchboard phrase *they're kind of in between right now*. Note vowel reduction in *they're* and *of*, coda deletion in *kind* and *right*, and resyllabification (the [v] of *of* attaches as the onset of *in*). Time is given in number of seconds from the beginning of sentence to the start of each syllable.

The Buckeye corpus (Pitt et al. 2007, Pitt et al. 2005) is a phonetically transcribed corpus of spontaneous American speech, containing about 300,000 words from 40 talkers. Phonetically transcribed corpora are also available for other languages, including the Kiel corpus of German and Mandarin corpora transcribed by the Chinese Academy of Social Sciences (Li et al., 2000).

In addition to resources like dictionaries and corpora, there are many useful phonetic software tools. One of the most versatile is the Praat package (Boersma and Weenink, 2005), which includes spectrum and spectrogram analysis, pitch extraction and formant analysis, and an embedded scripting language for automation.

27.7 Summary

This chapter has introduced many of the important concepts of phonetics and computational phonetics.

- We can represent the pronunciation of words in terms of units called **phones**. The standard system for representing phones is the **International Phonetic Alphabet** or **IPA**. The most common computational system for transcription of English is the **ARPAbet**, which conveniently uses ASCII symbols.
- Phones can be described by how they are produced **articulatorily** by the vocal organs; consonants are defined in terms of their **place** and **manner** of articulation and **voicing**; vowels by their **height**, **backness**, and **roundness**.
- A **phoneme** is a generalization or abstraction over different phonetic realizations. **Allophonic rules** express how a phoneme is realized in a given context.
- Speech sounds can also be described **acoustically**. Sound waves can be described in terms of **frequency**, **amplitude**, or their perceptual correlates, **pitch** and **loudness**.
- The **spectrum** of a sound describes its different frequency components. While some phonetic properties are recognizable from the waveform, both humans and machines rely on spectral analysis for phone detection.
- A **spectrogram** is a plot of a spectrum over time. Vowels are described by characteristic harmonics called **formants**.

- **Pronunciation dictionaries** are widely available and used for both speech recognition and synthesis, including the CMU dictionary for English and CELEX dictionaries for English, German, and Dutch. Other dictionaries are available from the LDC.
- Phonetically transcribed corpora are a useful resource for building computational models of phone variation and reduction in natural speech.

Bibliographical and Historical Notes

The major insights of articulatory phonetics date to the linguists of 800–150 B.C. India. They invented the concepts of place and manner of articulation, worked out the glottal mechanism of voicing, and understood the concept of assimilation. European science did not catch up with the Indian phoneticians until over 2000 years later, in the late 19th century. The Greeks did have some rudimentary phonetic knowledge; by the time of Plato's *Theaetetus* and *Cratylus*, for example, they distinguished vowels from consonants, and stop consonants from continuants. The Stoics developed the idea of the syllable and were aware of phonotactic constraints on possible words. An unknown Icelandic scholar of the 12th century exploited the concept of the phoneme and proposed a phonemic writing system for Icelandic, including diacritics for length and nasality. But his text remained unpublished until 1818 and even then was largely unknown outside Scandinavia (Robins, 1967). The modern era of phonetics is usually said to have begun with Sweet, who proposed what is essentially the phoneme in his *Handbook of Phonetics* (1877). He also devised an alphabet for transcription and distinguished between *broad* and *narrow* transcription, proposing many ideas that were eventually incorporated into the IPA. Sweet was considered the best practicing phonetician of his time; he made the first scientific recordings of languages for phonetic purposes and advanced the state of the art of articulatory description. He was also infamously difficult to get along with, a trait that is well captured in Henry Higgins, the stage character that George Bernard Shaw modeled after him. The phoneme was first named by the Polish scholar Baudouin de Courtenay, who published his theories in 1894.

Students with further interest in transcription and articulatory phonetics should consult an introductory phonetics textbook such as Ladefoged (1993) or Clark and Yallop (1995). Pullum and Ladusaw (1996) is a comprehensive guide to each of the symbols and diacritics of the IPA. A good resource for details about reduction and other phonetic processes in spoken English is Shockey (2003). Wells (1982) is the definitive three-volume source on dialects of English.

Many of the classic insights in acoustic phonetics had been developed by the late 1950s or early 1960s; just a few highlights include techniques like the sound spectrograph (Koenig et al., 1946), theoretical insights like the working out of the source-filter theory and other issues in the mapping between articulation and acoustics ((Fant, 1960), Stevens et al. 1953, Stevens and House 1955, Heinz and Stevens 1961, Stevens and House 1961) the F1xF2 space of vowel formants (Peterson and Barney, 1952), the understanding of the phonetic nature of stress and the use of duration and intensity as cues (Fry, 1955), and a basic understanding of issues in phone perception (Miller and Nicely 1955, Liberman et al. 1952). Lehiste (1967) is a collection of classic papers on acoustic phonetics. Many of the seminal papers of Gunnar Fant have been collected in Fant (2004).

Excellent textbooks on acoustic phonetics include Johnson (2003) and Ladefoged (1996). Coleman (2005) includes an introduction to computational processing

of acoustics as well as other speech processing issues, from a linguistic perspective. Stevens (1998) lays out an influential theory of speech sound production. A wide variety of books address speech from a signal processing and electrical engineering perspective. The ones with the greatest coverage of computational phonetics issues include Huang et al. (2001), O'Shaughnessy (2000), and Gold and Morgan (1999). Excellent textbooks on digital signal processing are Lyons (2004) and Rabiner and Schafer (1978).

There are a number of software packages for acoustic phonetic analysis. Probably the most widely-used one is **Praat** (Boersma and Weenink, 2005).

Many phonetics papers of computational interest are to be found in the *Journal of the Acoustical Society of America* (JASA), *Computer Speech and Language*, and *Speech Communication*.

Exercises

27.1 Find the mistakes in the ARPAbet transcriptions of the following words:

- | | | |
|----------------------|---------------------------------|------------------------|
| a. “three” [dh r i] | d. “study” [s t uh d i] | g. “slight” [s l iy t] |
| b. “sing” [s ih n g] | e. “though” [th ow] | |
| c. “eyes” [ay s] | f. “planning” [p pl aa n ih ng] | |

27.2 Translate the pronunciations of the following color words from the IPA into the ARPAbet (and make a note if you think you pronounce them differently than this!):

- | | | |
|-------------|--------------|-----------|
| a. [red] | e. [blæk] | i. [pjus] |
| b. [blu] | f. [wait] | j. [toup] |
| c. [grin] | g. ['ɔrindʒ] | |
| d. ['jelov] | h. ['pɜ:pł] | |

27.3 Ira Gershwin's lyric for *Let's Call the Whole Thing Off* talks about two pronunciations (each) of the words “tomato”, “potato”, and “either”. Transcribe into the ARPAbet both pronunciations of each of these three words.

27.4 Transcribe the following words in the ARPAbet:

1. dark
2. suit
3. greasy
4. wash
5. water

27.5 Take a wavefile of your choice. Some examples are on the textbook website. Download the Praat software, and use it to transcribe the wavefiles at the word level and into ARPAbet phones, using Praat to help you play pieces of each wavefile and to look at the wavefile and the spectrogram.

27.6 Record yourself saying five of the English vowels: [aa], [eh], [ae], [iy], [uw]. Find F1 and F2 for each of your vowels.

CHAPTER

28

Speech Recognition and Synthesis

Placeholder

Appendices

CHAPTER

A

Hidden Markov Models

Chapter 8 introduced the Hidden Markov Model and applied it to part of speech tagging. Part of speech tagging is a fully-supervised learning task, because we have a corpus of words labeled with the correct part-of-speech tag. But many applications don't have labeled data. So in this chapter, we introduce the full set of algorithms for HMMs, including the key unsupervised learning algorithm for HMM, the Forward-Backward algorithm. We'll repeat some of the text from Chapter 8 for readers who want the whole story laid out in a single chapter.

A.1 Markov Chains

Markov chain

The HMM is based on augmenting the Markov chain. A **Markov chain** is a model that tells us something about the probabilities of sequences of random variables, *states*, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, like the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. The states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.

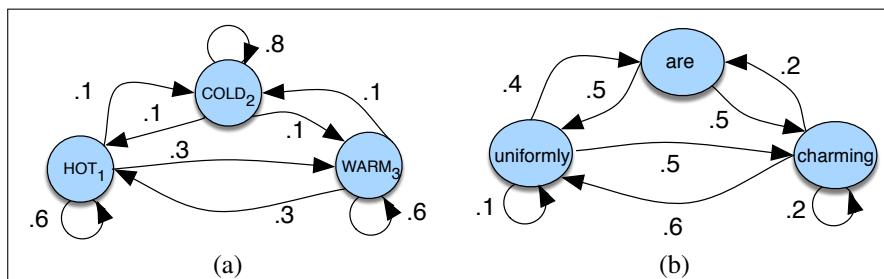


Figure A.1 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Markov assumption

More formally, consider a sequence of state variables q_1, q_2, \dots, q_i . A Markov model embodies the **Markov assumption** on the probabilities of this sequence: that when predicting the future, the past doesn't matter, only the present.

$$\text{Markov Assumption: } P(q_i = a | q_1 \dots q_{i-1}) = P(q_i = a | q_{i-1}) \quad (\text{A.1})$$

Figure A.1a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. The states are represented as nodes in the graph, and the transitions, with their probabilities, as edges. The transitions are probabilities: the values of arcs leaving a given

state must sum to 1. Figure A.1b shows a Markov chain for assigning a probability to a sequence of words $w_1 \dots w_n$. This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability $p(w_i|w_j)$! Given the two models in Fig. A.1, we can assign a probability to any sequence from our vocabulary.

Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N states
$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$	a transition probability matrix A , each a_{ij} representing the probability of moving from state i to state j , s.t. $\sum_{j=1}^n a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

Before you go on, use the sample probabilities in Fig. A.1a (with $\pi = [.1, .7, .2]$) to compute the probability of each of the following sequences:

- (A.2) hot hot hot hot
- (A.3) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. A.1a?

A.2 The Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags **hidden** because they are not observed.

Hidden Markov model A **hidden Markov model (HMM)** allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model. An HMM is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N states
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a transition probability matrix A , each a_{ij} representing the probability of moving from state i to state j , s.t. $\sum_{j=1}^N a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T observations , each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of observation likelihoods , also called emission probabilities , each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an initial probability distribution over states. π_i is the probability that the Markov chain will start in state i . Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^n \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends only on the previous state:

$$\text{Markov Assumption: } P(q_i|q_1 \dots q_{i-1}) = P(q_i|q_{i-1}) \quad (\text{A.4})$$

Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:

$$\text{Output Independence: } P(o_i|q_1 \dots q_i, \dots, q_T, o_1, \dots, o_i, \dots, o_T) = P(o_i|q_i) \quad (\text{A.5})$$

To exemplify these models, we'll use a task invented by Jason Eisner (2002). Imagine that you are a climatologist in the year 2799 studying the history of global warming. You cannot find any records of the weather in Baltimore, Maryland, for the summer of 2020, but you do find Jason Eisner's diary, which lists how many ice creams Jason ate every day that summer. Our goal is to use these observations to estimate the temperature every day. We'll simplify this weather task by assuming there are only two kinds of days: cold (C) and hot (H). So the Eisner task is as follows:

Given a sequence of observations O (each an integer representing the number of ice creams eaten on a given day) find the ‘hidden’ sequence Q of weather states (H or C) which caused Jason to eat the ice cream.

Figure A.2 shows a sample HMM for the ice cream task. The two hidden states (H and C) correspond to hot and cold weather, and the observations (drawn from the alphabet $O = \{1, 2, 3\}$) correspond to the number of ice creams eaten by Jason on a given day.



Figure A.2 A hidden Markov model for relating numbers of ice creams eaten by Jason (the observations) to the weather (H or C, the hidden variables).

An influential tutorial by Rabiner (1989), based on tutorials by Jack Ferguson in the 1960s, introduced the idea that hidden Markov models should be characterized by **three fundamental problems**:

- Problem 1 (Likelihood):** Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.
- Problem 2 (Decoding):** Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q .
- Problem 3 (Learning):** Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B .

We already saw an example of Problem 2 in Chapter 8. In the next two sections we introduce the Forward and Forward-Backward algorithms to solve Problems 1 and 3 and give more information on Problem 2.

A.3 Likelihood Computation: The Forward Algorithm

Our first problem is to compute the likelihood of a particular observation sequence. For example, given the ice-cream eating HMM in Fig. A.2, what is the probability of the sequence 3 1 3? More formally:

Computing Likelihood: Given an HMM $\lambda = (A, B)$ and an observation sequence O , determine the likelihood $P(O|\lambda)$.

For a Markov chain, where the surface observations are the same as the hidden events, we could compute the probability of 3 1 3 just by following the states labeled 3 1 3 and multiplying the probabilities along the arcs. For a hidden Markov model, things are not so simple. We want to determine the probability of an ice-cream observation sequence like 3 1 3, but we don't know what the hidden state sequence is!

Let's start with a slightly simpler situation. Suppose we already knew the weather and wanted to predict how much ice cream Jason would eat. This is a useful part of many HMM tasks. For a given hidden state sequence (e.g., *hot hot cold*), we can easily compute the output likelihood of 3 1 3.

Let's see how. First, recall that for hidden Markov models, each hidden state produces only a single observation. Thus, the sequence of hidden states and the sequence of observations have the same length.¹

Given this one-to-one mapping and the Markov assumptions expressed in Eq. A.4, for a particular hidden state sequence $Q = q_0, q_1, q_2, \dots, q_T$ and an observation sequence $O = o_1, o_2, \dots, o_T$, the likelihood of the observation sequence is

$$P(O|Q) = \prod_{i=1}^T P(o_i|q_i) \quad (\text{A.6})$$

The computation of the forward probability for our ice-cream observation 3 1 3 from one possible hidden state sequence *hot hot cold* is shown in Eq. A.7. Figure A.3 shows a graphic representation of this computation.

$$P(3\ 1\ 3|\text{hot hot cold}) = P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \quad (\text{A.7})$$

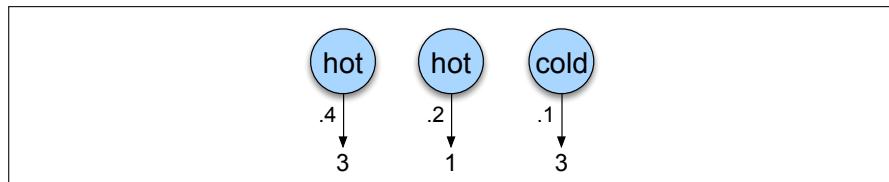


Figure A.3 The computation of the observation likelihood for the ice-cream events 3 1 3 given the hidden state sequence *hot hot cold*.

But of course, we don't actually know what the hidden state (weather) sequence was. We'll need to compute the probability of ice-cream events 3 1 3 instead by

¹ In a variant of HMMs called **segmental HMMs** (in speech recognition) or **semi-HMMs** (in text processing) this one-to-one mapping between the length of the hidden state sequence and the length of the observation sequence does not hold.

summing over all possible weather sequences, weighted by their probability. First, let's compute the joint probability of being in a particular weather sequence Q and generating a particular sequence O of ice-cream events. In general, this is

$$P(O, Q) = P(O|Q) \times P(Q) = \prod_{i=1}^T P(o_i|q_i) \times \prod_{i=1}^T P(q_i|q_{i-1}) \quad (\text{A.8})$$

The computation of the joint probability of our ice-cream observation $3 \ 1 \ 3$ and one possible hidden state sequence *hot hot cold* is shown in Eq. A.9. Figure A.4 shows a graphic representation of this computation.

$$\begin{aligned} P(3 \ 1 \ 3, \text{hot hot cold}) &= P(\text{hot|start}) \times P(\text{hot|hot}) \times P(\text{cold|hot}) \\ &\quad \times P(3|\text{hot}) \times P(1|\text{hot}) \times P(3|\text{cold}) \end{aligned} \quad (\text{A.9})$$



Figure A.4 The computation of the joint probability of the ice-cream events $3 \ 1 \ 3$ and the hidden state sequence *hot hot cold*.

Now that we know how to compute the joint probability of the observations with a particular hidden state sequence, we can compute the total probability of the observations just by summing over all possible hidden state sequences:

$$P(O) = \sum_Q P(O, Q) = \sum_Q P(O|Q)P(Q) \quad (\text{A.10})$$

For our particular case, we would sum over the eight 3-event sequences *cold cold cold*, *cold cold hot*, that is,

$$P(3 \ 1 \ 3) = P(3 \ 1 \ 3, \text{cold cold cold}) + P(3 \ 1 \ 3, \text{cold cold hot}) + P(3 \ 1 \ 3, \text{hot hot cold}) + \dots$$

For an HMM with N hidden states and an observation sequence of T observations, there are N^T possible hidden sequences. For real tasks, where N and T are both large, N^T is a very large number, so we cannot compute the total observation likelihood by computing a separate observation likelihood for each hidden state sequence and then summing them.

forward
algorithm

Instead of using such an extremely exponential algorithm, we use an efficient $O(N^2 T)$ algorithm called the **forward algorithm**. The forward algorithm is a kind of **dynamic programming** algorithm, that is, an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does so efficiently by implicitly folding each of these paths into a single **forward trellis**.

Figure A.5 shows an example of the forward trellis for computing the likelihood of $3 \ 1 \ 3$ given the hidden state sequence *hot hot cold*.



Figure A.5 The forward trellis for computing the total observation likelihood for the ice-cream events 3 1 3. Hidden states are in circles, observations in squares. The figure shows the computation of $\alpha_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.12: $\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.11: $\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda)$.

Each cell of the forward algorithm trellis $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations, given the automaton λ . The value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of every path that could lead us to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2 \dots o_t, q_t = j | \lambda) \quad (\text{A.11})$$

Here, $q_t = j$ means “the t^{th} state in the sequence of states is state j ”. We compute this probability $\alpha_t(j)$ by summing over the extensions of all the paths that lead to the current cell. For a given state q_j at time t , the value $\alpha_t(j)$ is computed as

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t) \quad (\text{A.12})$$

The three factors that are multiplied in Eq. A.12 in extending the previous paths to compute the forward probability at time t are

$\alpha_{t-1}(i)$	the previous forward path probability from the previous time step
a_{ij}	the transition probability from previous state q_i to current state q_j
$b_j(o_t)$	the state observation likelihood of the observation symbol o_t given the current state j

Consider the computation in Fig. A.5 of $\alpha_2(2)$, the forward probability of being at time step 2 in state 2 having generated the partial observation 3 1. We compute by extending the α probabilities from time step 1, via two paths, each extension consisting of the three factors above: $\alpha_1(1) \times P(H|C) \times P(1|H)$ and $\alpha_1(2) \times P(H|H) \times P(1|H)$.

Figure A.6 shows another visualization of this induction step for computing the value in one new cell of the trellis.

We give two formal definitions of the forward algorithm: the pseudocode in Fig. A.7 and a statement of the definitional recursion here.

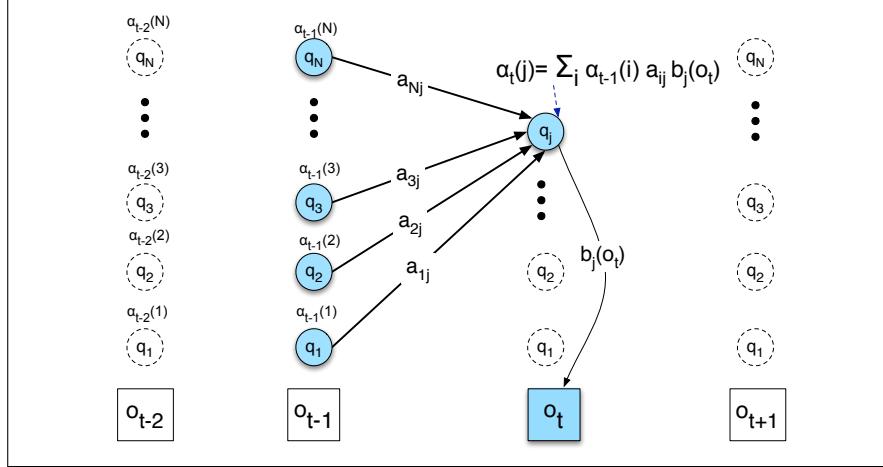


Figure A.6 Visualizing the computation of a single element $\alpha_t(i)$ in the trellis by summing all the previous values α_{t-1} , weighted by their transition probabilities a , and multiplying by the observation probability $b_i(o_t)$. For many applications of HMMs, many of the transition probabilities are 0, so not all previous states will contribute to the forward probability of the current state. Hidden states are in circles, observations in squares. Shaded nodes are included in the probability computation for $\alpha_t(i)$.

```

function FORWARD(observations of len  $T$ , state-graph of len  $N$ ) returns forward-prob
    create a probability matrix forward[ $N, T$ ]
    for each state  $s$  from 1 to  $N$  do ; initialization step
        forward[ $s, 1$ ]  $\leftarrow \pi_s * b_s(o_1)
    for each time step  $t$  from 2 to  $T$  do ; recursion step
        for each state  $s$  from 1 to  $N$  do
            forward[ $s, t$ ]  $\leftarrow \sum_{s'=1}^N$  forward[ $s', t - 1$ ] *  $a_{s', s} * b_s(o_t)$ 
    forwardprob  $\leftarrow \sum_{s=1}^N$  forward[ $s, T$ ] ; termination step
    return forwardprob$ 
```

Figure A.7 The forward algorithm, where $forward[s, t]$ represents $\alpha_t(s)$.

1. Initialization:

$$\alpha_1(j) = \pi_j b_j(o_1) \quad 1 \leq j \leq N$$

2. Recursion:

$$\alpha_t(j) = \sum_{i=1}^N \alpha_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$

3. Termination:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i)$$

A.4 Decoding: The Viterbi Algorithm

For any model, such as an HMM, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of observations is called the **decoding** task. In the ice-cream domain, given a sequence of ice-cream observations 3 1 3 and an HMM, the task of the **decoder** is to find the best hidden weather sequence (H H H). More formally,

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, \dots, o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \dots q_T$.

We might propose to find the best sequence as follows: For each possible hidden state sequence (HHH, HHC, HCH, etc.), we could run the forward algorithm and compute the likelihood of the observation sequence given that hidden state sequence. Then we could choose the hidden state sequence with the maximum observation likelihood. It should be clear from the previous section that we cannot do this because there are an exponentially large number of state sequences.

Instead, the most common decoding algorithms for HMMs is the **Viterbi algorithm**. Like the forward algorithm, Viterbi is a kind of **dynamic programming** that makes uses of a dynamic programming trellis. Viterbi also strongly resembles another dynamic programming variant, the **minimum edit distance** algorithm of Chapter 2.



Figure A.8 The Viterbi trellis for computing the best path through the hidden state space for the ice-cream eating events 3 1 3. Hidden states are in circles, observations in squares. White (unfilled) circles indicate illegal transitions. The figure shows the computation of $v_t(j)$ for two states at two time steps. The computation in each cell follows Eq. A.14: $v_t(j) = \max_{1 \leq i \leq N-1} v_{t-1}(i) a_{ij} b_j(o_t)$. The resulting probability expressed in each cell is Eq. A.13: $v_t(j) = P(q_0, q_1, \dots, q_{t-1}, o_1, o_2, \dots, o_t, q_t = j | \lambda)$.

Figure A.8 shows an example of the Viterbi trellis for computing the best hidden state sequence for the observation sequence 3 1 3. The idea is to process the observation sequence left to right, filling out the trellis. Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence q_1, \dots, q_{t-1} , given the

automaton λ . The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability

$$v_t(j) = \max_{q_1, \dots, q_{t-1}} P(q_1 \dots q_{t-1}, o_1, o_2 \dots o_t, q_t = j | \lambda) \quad (\text{A.13})$$

Note that we represent the most probable path by taking the maximum over all possible previous state sequences $\max_{q_1, \dots, q_{t-1}}$. Like other dynamic programming algorithms, Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t - 1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t , the value $v_t(j)$ is computed as

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) \quad (\text{A.14})$$

The three factors that are multiplied in Eq. A.14 for extending the previous paths to compute the Viterbi probability at time t are

$v_{t-1}(i)$	the previous Viterbi path probability from the previous time step
a_{ij}	the transition probability from previous state q_i to current state q_j
$b_j(o_t)$	the state observation likelihood of the observation symbol o_t given the current state j

```

function VITERBI(observations of len  $T$ , state-graph of len  $N$ ) returns best-path, path-prob

create a path probability matrix viterbi[ $N, T$ ]
for each state  $s$  from 1 to  $N$  do ; initialization step
    viterbi[ $s, 1$ ]  $\leftarrow \pi_s * b_s(o_1)$ 
    backpointer[ $s, 1$ ]  $\leftarrow 0$ 
for each time step  $t$  from 2 to  $T$  do ; recursion step
    for each state  $s$  from 1 to  $N$  do
        viterbi[ $s, t$ ]  $\leftarrow \max_{s'=1}^N$  viterbi[ $s', t - 1$ ] *  $a_{s', s} * b_s(o_t)$ 
        backpointer[ $s, t$ ]  $\leftarrow \operatorname{argmax}_{s'=1}^N$  viterbi[ $s', t - 1$ ] *  $a_{s', s} * b_s(o_t)$ 
    bestpathprob  $\leftarrow \max_{s=1}^N$  viterbi[ $s, T$ ] ; termination step
    bestpathpointer  $\leftarrow \operatorname{argmax}_{s=1}^N$  viterbi[ $s, T$ ] ; termination step
    bestpath  $\leftarrow$  the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob

```

Figure A.9 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A, B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

Figure A.9 shows pseudocode for the Viterbi algorithm. Note that the Viterbi algorithm is identical to the forward algorithm except that it takes the **max** over the previous path probabilities whereas the forward algorithm takes the **sum**. Note also that the Viterbi algorithm has one component that the forward algorithm doesn't

have: **backpointers**. The reason is that while the forward algorithm needs to produce an observation likelihood, the Viterbi algorithm must produce a probability and also the most likely state sequence. We compute this best state sequence by keeping track of the path of hidden states that led to each state, as suggested in Fig. A.10, and then at the end backtracing the best path to the beginning (the Viterbi **backtrace**).



Figure A.10 The Viterbi backtrace. As we extend each path to a new state account for the next observation, we keep a backpointer (shown with broken lines) to the best path that led us to this state.

Finally, we can give a formal definition of the Viterbi recursion as follows:

1. **Initialization:**

$$\begin{aligned} v_1(j) &= \pi_j b_j(o_1) & 1 \leq j \leq N \\ bt_1(j) &= 0 & 1 \leq j \leq N \end{aligned}$$

2. **Recursion**

$$\begin{aligned} v_t(j) &= \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); & 1 \leq j \leq N, 1 < t \leq T \\ bt_t(j) &= \operatorname{argmax}_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t); & 1 \leq j \leq N, 1 < t \leq T \end{aligned}$$

3. **Termination:**

$$\text{The best score: } P_* = \max_{i=1}^N v_T(i)$$

$$\text{The start of backtrace: } qt_* = \operatorname{argmax}_{i=1}^N v_T(i)$$

A.5 HMM Training: The Forward-Backward Algorithm

We turn to the third problem for HMMs: learning the parameters of an HMM, that is, the A and B matrices. Formally,

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B .

The input to such a learning algorithm would be an unlabeled sequence of observations O and a vocabulary of potential hidden states Q . Thus, for the ice cream task, we would start with a sequence of observations $O = \{1, 3, 2, \dots\}$ and the set of hidden states H and C .

**Forward-backward
Baum-Welch
EM**

The standard algorithm for HMM training is the **forward-backward**, or **Baum-Welch** algorithm (Baum, 1972), a special case of the **Expectation-Maximization** or EM algorithm (Dempster et al., 1977). The algorithm will let us train both the transition probabilities A and the emission probabilities B of the HMM. EM is an *iterative* algorithm, computing an initial estimate for the probabilities, then using those estimates to computing a better estimate, and so on, iteratively improving the probabilities that it learns.

Let us begin by considering the much simpler case of training a fully visible Markov model, we're know both the temperature and the ice cream count for every day. That is, imagine we see the following set of input observations and magically knew the aligned hidden state sequences:

3	3	2	1	1	2	1	2	3
hot	hot	cold	cold	cold	cold	cold	hot	hot

This would easily allow us to compute the HMM parameters just by maximum likelihood estimation from the training data. First, we can compute π from the count of the 3 initial hidden states:

$$\pi_h = 1/3 \quad \pi_c = 2/3$$

Next we can directly compute the A matrix from the transitions, ignoring the final hidden states:

$$\begin{aligned} p(\text{hot}|\text{hot}) &= 2/3 & p(\text{cold}|\text{hot}) &= 1/3 \\ p(\text{cold}|\text{cold}) &= 2/3 & p(\text{hot}|\text{cold}) &= 1/3 \end{aligned}$$

and the B matrix:

$$\begin{aligned} P(1|\text{hot}) &= 0/4 = 0 & P(1|\text{cold}) &= 3/5 = .6 \\ P(2|\text{hot}) &= 1/4 = .25 & P(2|\text{cold}) &= 2/5 = .4 \\ P(3|\text{hot}) &= 3/4 = .75 & P(3|\text{cold}) &= 0 \end{aligned}$$

For a real HMM, we cannot compute these counts directly from an observation sequence since we don't know which path of states was taken through the machine for a given input. For example, suppose I didn't tell you the temperature on day 2, and you had to guess it, but you (magically) had the above probabilities, and the temperatures on the other days. You could do some Bayesian arithmetic with all the other probabilities to get estimates of the likely temperature on that missing day, and use those to get expected counts for the temperatures for day 2.

But the real problem is even harder: we don't know the counts of being in **any** of the hidden states!! The Baum-Welch algorithm solves this by *iteratively* estimating the counts. We will start with an estimate for the transition and observation probabilities and then use these estimated probabilities to derive better and better probabilities. And we're going to do this by computing the forward probability for an observation and then dividing that probability mass among all the different paths that contributed to this forward probability.

backward probability

To understand the algorithm, we need to define a useful probability related to the forward probability and called the **backward probability**. The backward probabil-

ity β is the probability of seeing the observations from time $t + 1$ to the end, given that we are in state i at time t (and given the automaton λ):

$$\beta_t(i) = P(o_{t+1}, o_{t+2} \dots o_T | q_t = i, \lambda) \quad (\text{A.15})$$

It is computed inductively in a similar manner to the forward algorithm.

1. Initialization:

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

2. Recursion

$$\beta_t(i) = \sum_{j=1}^N a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, 1 \leq t < T$$

3. Termination:

$$P(O|\lambda) = \sum_{j=1}^N \pi_j b_j(o_1) \beta_1(j)$$

Figure A.11 illustrates the backward induction step.



Figure A.11 The computation of $\beta_t(i)$ by summing all the successive values $\beta_{t+1}(j)$ weighted by their transition probabilities a_{ij} and their observation probabilities $b_j(o_{t+1})$. Start and end states not shown.

We are now ready to see how the forward and backward probabilities can help compute the transition probability a_{ij} and observation probability $b_i(o_t)$ from an observation sequence, even though the actual path taken through the model is hidden.

Let's begin by seeing how to estimate \hat{a}_{ij} by a variant of simple maximum likelihood estimation:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \quad (\text{A.16})$$

How do we compute the numerator? Here's the intuition. Assume we had some estimate of the probability that a given transition $i \rightarrow j$ was taken at a particular point in time t in the observation sequence. If we knew this probability for each

particular time t , we could sum over all times t to estimate the total count for the transition $i \rightarrow j$.

More formally, let's define the probability ξ_t as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence and of course the model:

$$\xi_t(i, j) = P(q_t = i, q_{t+1} = j | O, \lambda) \quad (\text{A.17})$$

To compute ξ_t , we first compute a probability which is similar to ξ_t , but differs in including the probability of the observation; note the different conditioning of O from Eq. A.17:

$$\text{not-quite-}\xi_t(i, j) = P(q_t = i, q_{t+1} = j, O | \lambda) \quad (\text{A.18})$$



Figure A.12 Computation of the joint probability of being in state i at time t and state j at time $t+1$. The figure shows the various probabilities that need to be combined to produce $P(q_t = i, q_{t+1} = j, O | \lambda)$: the α and β probabilities, the transition probability a_{ij} and the observation probability $b_j(o_{t+1})$. After Rabiner (1989) which is ©1989 IEEE.

Figure A.12 shows the various probabilities that go into computing not-quite- ξ_t : the transition probability for the arc in question, the α probability before the arc, the β probability after the arc, and the observation probability for the symbol just after the arc. These four are multiplied together to produce *not-quite-* ξ_t as follows:

$$\text{not-quite-}\xi_t(i, j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \quad (\text{A.19})$$

To compute ξ_t from *not-quite-* ξ_t , we follow the laws of probability and divide by $P(O | \lambda)$, since

$$P(X|Y, Z) = \frac{P(X, Y|Z)}{P(Y|Z)} \quad (\text{A.20})$$

The probability of the observation given the model is simply the forward probability of the whole utterance (or alternatively, the backward probability of the whole utterance):

$$P(O | \lambda) = \sum_{j=1}^N \alpha_t(j) \beta_t(j) \quad (\text{A.21})$$

So, the final equation for ξ_t is

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^N \alpha_t(j) \beta_t(j)} \quad (\text{A.22})$$

The expected number of transitions from state i to state j is then the sum over all t of ξ . For our estimate of a_{ij} in Eq. A.16, we just need one more thing: the total expected number of transitions from state i . We can get this by summing over all transitions out of state i . Here's the final formula for \hat{a}_{ij} :

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^N \xi_t(i, k)} \quad (\text{A.23})$$

We also need a formula for recomputing the observation probability. This is the probability of a given symbol v_k from the observation vocabulary V , given a state j : $\hat{b}_j(v_k)$. We will do this by trying to compute

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \quad (\text{A.24})$$

For this, we will need to know the probability of being in state j at time t , which we will call $\gamma_t(j)$:

$$\gamma_t(j) = P(q_t = j | O, \lambda) \quad (\text{A.25})$$

Once again, we will compute this by including the observation sequence in the probability:

$$\gamma_t(j) = \frac{P(q_t = j, O | \lambda)}{P(O | \lambda)} \quad (\text{A.26})$$



Figure A.13 The computation of $\gamma_t(j)$, the probability of being in state j at time t . Note that γ is really a degenerate case of ξ and hence this figure is like a version of Fig. A.12 with state i collapsed with state j . After Rabiner (1989) which is ©1989 IEEE.

As Fig. A.13 shows, the numerator of Eq. A.26 is just the product of the forward probability and the backward probability:

$$\gamma_t(j) = \frac{\alpha_t(j) \beta_t(j)}{P(O | \lambda)} \quad (\text{A.27})$$

We are ready to compute b . For the numerator, we sum $\gamma_t(j)$ for all time steps t in which the observation o_t is the symbol v_k that we are interested in. For the denominator, we sum $\gamma_t(j)$ over all time steps t . The result is the percentage of the times that we were in state j and saw symbol v_k (the notation $\sum_{t=1}^T s.t. O_t=v_k$ means “sum over all t for which the observation at time t was v_k ”):

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^T s.t. O_t=v_k \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)} \quad (\text{A.28})$$

We now have ways in Eq. A.23 and Eq. A.28 to *re-estimate* the transition A and observation B probabilities from an observation sequence O , assuming that we already have a previous estimate of A and B .

These re-estimations form the core of the iterative forward-backward algorithm. The forward-backward algorithm (Fig. A.14) starts with some initial estimate of the HMM parameters $\lambda = (A, B)$. We then iteratively run two steps. Like other cases of the EM (expectation-maximization) algorithm, the forward-backward algorithm has two steps: the **expectation step**, or **E-step**, and the **maximization step**, or **M-step**.

E-step
M-step In the E-step, we compute the expected state occupancy count γ and the expected state transition count ξ from the earlier A and B probabilities. In the M-step, we use γ and ξ to recompute new A and B probabilities.

```

function FORWARD-BACKWARD(observations of len  $T$ , output vocabulary  $V$ , hidden state set  $Q$ ) returns HMM=( $A, B$ )
    initialize  $A$  and  $B$ 
    iterate until convergence
        E-step
             $\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall t \text{ and } j$ 
             $\xi_t(i, j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall t, i, \text{ and } j$ 
        M-step
            
$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \sum_{k=1}^N \xi_t(i, k)}$$

            
$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^T s.t. O_t=v_k \gamma_t(j)}{\sum_{t=1}^T \gamma_t(j)}$$

    return  $A, B$ 

```

Figure A.14 The forward-backward algorithm.

Although in principle the forward-backward algorithm can do completely unsupervised learning of the A and B parameters, in practice the initial conditions are very important. For this reason the algorithm is often given extra information. For example, for HMM-based speech recognition, the HMM structure is often set by hand, and only the emission (B) and (non-zero) A transition probabilities are trained from a set of observation sequences O .

A.6 Summary

This chapter introduced the **hidden Markov model** for probabilistic **sequence classification**.

- Hidden Markov models (**HMMs**) are a way of relating a sequence of **observations** to a sequence of **hidden classes** or **hidden states** that explain the observations.
- The process of discovering the sequence of hidden states, given the sequence of observations, is known as **decoding** or **inference**. The **Viterbi** algorithm is commonly used for decoding.
- The parameters of an HMM are the A transition probability matrix and the B observation likelihood matrix. Both can be trained with the **Baum-Welch** or **forward-backward** algorithm.

Bibliographical and Historical Notes

As we discussed in Chapter 8, Markov chains were first used by [Markov \(1913\)](#) (translation [Markov 2006](#)), to predict whether an upcoming letter in Pushkin’s *Eugene Onegin* would be a vowel or a consonant. The hidden Markov model was developed by Baum and colleagues at the Institute for Defense Analyses in Princeton ([Baum and Petrie 1966](#), [Baum and Eagon 1967](#)).

The **Viterbi** algorithm was first applied to speech and language processing in the context of speech recognition by [Vintsyuk \(1968\)](#) but has what [Kruskal \(1983\)](#) calls a “remarkable history of multiple independent discovery and publication”. Kruskal and others give at least the following independently-discovered variants of the algorithm published in four separate fields:

Citation	Field
Viterbi (1967)	information theory
Vintsyuk (1968)	speech processing
Needleman and Wunsch (1970)	molecular biology
Sakoe and Chiba (1971)	speech processing
Sankoff (1972)	molecular biology
Reichert et al. (1973)	molecular biology
Wagner and Fischer (1974)	computer science

The use of the term **Viterbi** is now standard for the application of dynamic programming to any kind of probabilistic maximization problem in speech and language processing. For non-probabilistic problems (such as for minimum edit distance), the plain term **dynamic programming** is often used. [Forney, Jr. \(1973\)](#) wrote an early survey paper that explores the origin of the Viterbi algorithm in the context of information and communications theory.

Our presentation of the idea that hidden Markov models should be characterized by three fundamental problems was modeled after an influential tutorial by [Rabiner \(1989\)](#), which was itself based on tutorials by Jack Ferguson of IDA in the 1960s. [Jelinek \(1997\)](#) and [Rabiner and Juang \(1993\)](#) give very complete descriptions of the forward-backward algorithm as applied to the speech recognition problem. [Jelinek \(1997\)](#) also shows the relationship between forward-backward and EM.

Bibliography

- Abadi**, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.. Software available from tensorflow.org.
- Abney**, S. P. (1991). Parsing by chunks. In Berwick, R. C., Abney, S. P., and Tenny, C. (Eds.), *Principle-Based Parsing: Computation and Psycholinguistics*, 257–278. Kluwer.
- Abney**, S. P., Schapire, R. E., and Singer, Y. (1999). Boosting applied to tagging and PP attachment. In *EMNLP/VLC-99*, 38–45.
- Agarwal**, O., Subramanian, S., Nenкова, A., and Roth, D. (2019). Evaluation of named entity coreference. In *Workshop on Computational Models of Reference, Anaphora and Coreference*, 1–7.
- Aggarwal**, C. C. and Zhai, C. (2012). A survey of text classification algorithms. In Aggarwal, C. C. and Zhai, C. (Eds.), *Mining text data*, 163–222. Springer.
- Agichtein**, E. and Gravano, L. (2000). Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*.
- Agirre**, E. and de Lacalle, O. L. (2003). Clustering WordNet word senses. In *RANLP 2003*.
- Agirre**, E., Banea, C., Cardie, C., Cer, D., Diab, M., Gonzalez-Agirre, A., Guo, W., Lopez-Gazpio, I., Maritxalar, M., Mihalcea, R., Rigau, G., Uria, L., and Wiebe, J. (2015). 2015 SemEval-2015 Task 2: Semantic Textual Similarity, English, Spanish and Pilot on Interpretability. In *SemEval-15*, 252–263.
- Agirre**, E., Diab, M., Cer, D., and Gonzalez-Agirre, A. (2012). SemEval-2012 task 6: A pilot on semantic textual similarity. In *SemEval-12*, 385–393.
- Agirre**, E. and Edmonds, P. (Eds.). (2006). *Word Sense Disambiguation: Algorithms and Applications*. Kluwer.
- Agirre**, E. and Martinez, D. (2001). Learning class-to-class selectional preferences. In *CoNLL-01*.
- Aho**, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley.
- Aho**, A. V. and Ullman, J. D. (1972). *The Theory of Parsing, Translation, and Compiling*. Vol. 1. Prentice Hall.
- Ajdukiewicz**, K. (1935). Die syntaktische Konnektivität. *Studia Philosophica*, 1, 1–27. English translation “Syntactic Connexion” by H. Weber in McCall, S. (Ed.) 1967. *Polish Logic*, pp. 207–231, Oxford University Press.
- Algoet**, P. H. and Cover, T. M. (1988). A sandwich proof of the Shannon-McMillan-Breiman theorem. *The Annals of Probability*, 16(2), 899–909.
- Allen**, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23(2), 123–154.
- Allen**, J. and Perrault, C. R. (1980). Analyzing intention in utterances. *Artificial Intelligence*, 15, 143–178.
- Althoff**, T., Danescu-Niculescu-Mizil, C., and Jurafsky, D. (2014). How to ask for a favor: A case study on the success of altruistic requests. In *ICWSM 2014*.
- Amsler**, R. A. (1981). A taxonomy of English nouns and verbs. In *ACL-81*, 133–138.
- An**, J., Kwak, H., and Ahn, Y.-Y. (2018). SemAxis: A lightweight framework to characterize domain-specific word semantics beyond sentiment. In *ACL 2018*.
- Aone**, C. and Bennett, S. W. (1995). Evaluating automated and manual acquisition of anaphora resolution strategies. In *ACL-95*, 122–129.
- Ariel**, M. (2001). Accessibility theory: An overview. In Sanders, T., Schipperijn, J., and Spooren, W. (Eds.), *Text Representation: Linguistic and Psycholinguistic Aspects*, 29–87. Benjamins.
- Artstein**, R., Gandhi, S., Gerten, J., Leuski, A., and Traum, D. (2009). Semi-formal evaluation of conversational characters. In *Languages: From Formal to Natural*, 22–35. Springer.
- Asher**, N. (1993). *Reference to Abstract Objects in Discourse*. Studies in Linguistics and Philosophy (SLAP) 50, Kluwer.
- Asher**, N. and Lascarides, A. (2003). *Logics of Conversation*. Cambridge University Press.
- Austin**, J. L. (1962). *How to Do Things with Words*. Harvard University Press.
- Awadallah**, A. H., Kulkarni, R. G., Ozertem, U., and Jones, R. (2015). Charaterizing and predicting voice query reformulation. In *CIKM-15*.
- Baayen**, R. H. (2001). *Word frequency distributions*. Springer.
- Baayen**, R. H., Piepenbrock, R., and Gulikers, L. (1995). *The CELEX Lexical Database (Release 2) [CD-ROM]*. Linguistic Data Consortium, University of Pennsylvania [Distributor].
- Baccianella**, S., Esuli, A., and Sebastiani, F. (2010). Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining.. In *LREC-10*, 2200–2204.
- Bach**, K. and Harnish, R. (1979). *Linguistic communication and speech acts*. MIT Press.
- Backus**, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. In *Information Processing: Proceedings of the International Conference on Information Processing, Paris*, 125–132. UNESCO.
- Backus**, J. W. (1996). Transcript of question and answer session. In Wexelblat, R. L. (Ed.), *History of Programming Languages*, p. 162. Academic Press.
- Bagga**, A. and Baldwin, B. (1998). Algorithms for scoring coreference chains. In *LREC-98*, 563–566.
- Bahdanau**, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- Bahl**, L. R. and Mercer, R. L. (1976). Part of speech assignment by a statistical decision algorithm. In *Proceedings IEEE International Symposium on Information Theory*, 88–89.
- Bahl**, L. R., Jelinek, F., and Mercer, R. L. (1983). A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2), 179–190.
- Baker**, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet project. In *COLING/ACL-98*, 86–90.
- Baker**, J. K. (1975a). The DRAGON system – An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-23(1), 24–29.

- Baker, J. K.** (1975b). Stochastic modeling for automatic speech understanding. In Reddy, D. R. (Ed.), *Speech Recognition*. Academic Press.
- Baker, J. K.** (1979). Trainable grammars for speech recognition. In Klatt, D. H. and Wolf, J. J. (Eds.), *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, 547–550.
- Baldridge, J.**, Asher, N., and Hunter, J. (2007). Annotation for and robust parsing of discourse structure on unrestricted texts. *Zeitschrift für Sprachwissenschaft*, 26, 213–239.
- Bamman, D.**, O'Connor, B., and Smith, N. A. (2013). Learning latent personas of film characters. In *ACL 2013*, 352–361.
- Banarescu, L.**, Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 178–186.
- Bangalore, S.** and Joshi, A. K. (1999). Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2), 237–265.
- Banko, M.**, Cafarella, M., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction for the web. In *IJCAI*, 2670–2676.
- Bar-Hillel, Y.** (1953). A quasi-arithmetical notation for syntactic description. *Language*, 29, 47–58.
- Barker, C.** (2010). Nominals don't provide criteria of identity. In Rathert, M. and Alexiadou, A. (Eds.), *The Semantics of Nominalizations across Languages and Frameworks*, 9–24. Mouton de Gruyter, Berlin.
- Barzilay, R.** and Lapata, M. (2005). Modeling local coherence: An entity-based approach. In *ACL-05*, 141–148.
- Barzilay, R.** and Lapata, M. (2008). Modeling local coherence: An entity-based approach. *Computational Linguistics*, 34(1), 1–34.
- Barzilay, R.** and Lee, L. (2004). Catching the drift: Probabilistic content models, with applications to generation and summarization. In *HLT-NAACL-04*, 113–120.
- Basile, P.**, Caputo, A., and Semeraro, G. (2014). An enhanced Lesk word sense disambiguation algorithm through a distributional semantic model. In *COLING-14*, 1591–1600.
- Baum, L. E.** (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In Shisha, O. (Ed.), *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*, 1–8. Academic Press.
- Baum, L. E.** and Eagon, J. A. (1967). An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73(3), 360–363.
- Baum, L. E.** and Petrie, T. (1966). Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics*, 37(6), 1554–1563.
- Baum, L. F.** (1900). *The Wizard of Oz*. Available at Project Gutenberg.
- Bayes, T.** (1763). *An Essay Toward Solving a Problem in the Doctrine of Chances*, Vol. 53. Reprinted in *Facsimiles of Two Papers by Bayes*, Hafner Publishing, 1963.
- Bazell, C. E.** (1952/1966). The correspondence fallacy in structural linguistics. In Hamp, E. P., Householder, F. W., and Austerlitz, R. (Eds.), *Studies by Members of the English Department, Istanbul University* (3), reprinted in *Readings in Linguistics II* (1966), 271–298. University of Chicago Press.
- Bean, D.** and Riloff, E. (1999). Corpus-based identification of non-anaphoric noun phrases. In *ACL-99*, 373–380.
- Bean, D.** and Riloff, E. (2004). Unsupervised learning of contextual role knowledge for coreference resolution. In *HLT-NAACL-04*.
- Beckman, M. E.** and Ayers, G. M. (1997). Guidelines for ToBI labelling. Unpublished manuscript, Ohio State University, http://www.ling.ohio-state.edu/research/photonics/E_ToBI/.
- Beckman, M. E.** and Hirschberg, J. (1994). The ToBI annotation conventions. Manuscript, Ohio State University.
- Bedi, G.**, Carrillo, F., Cecchi, G. A., Slezak, D. F., Sigman, M., Mota, N. B., Ribeiro, S., Javitt, D. C., Copelli, M., and Corcoran, C. M. (2015). Automated analysis of free speech predicts psychosis onset in high-risk youths. *npj Schizophrenia*, 1.
- Bejček, E.**, Hajičová, E., Hajič, J., Jírová, P., Kettnerová, V., Kolářová, V., Mikulová, M., Mirovský, J., Nedoluzhko, A., Panevová, J., Poláková, L., Ševčíková, M., Štěpánek, J., and Zikánová, Š. (2013). Prague dependency treebank 3.0. Tech. rep., Institute of Formal and Applied Linguistics, Charles University in Prague. LINDAT/CLARIN digital library at Institute of Formal and Applied Linguistics, Charles University in Prague.
- Bellelgarda, J. R.** (1997). A latent semantic analysis framework for large-span language modeling. In *Eurospeech-97*.
- Bellelgarda, J. R.** (2000). Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 89(8), 1279–1296.
- Bellelgarda, J. R.** (2013). Natural language technology in mobile devices: Two grounding frameworks. In *Mobile Speech and Advanced Natural Language Solutions*, 185–196. Springer.
- Bellman, R.** (1957). *Dynamic Programming*. Princeton University Press.
- Bellman, R.** (1984). *Eye of the Hurricane: an autobiography*. World Scientific Singapore.
- Bengio, Y.**, Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798–1828.
- Bengio, Y.**, Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb), 1137–1155.
- Bengio, Y.**, Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *NIPS 2007*, 153–160.
- Bengio, Y.**, Schwenk, H., Senécal, J.-S., Morin, F., and Gauvain, J.-L. (2006). Neural probabilistic language models. In *Innovations in Machine Learning*, 137–186. Springer.
- Bentgson, E.** and Roth, D. (2008). Understanding the value of features for coreference resolution. In *EMNLP-08*, 294–303.
- Berant, J.**, Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *EMNLP 2013*.
- Berant, J.** and Liang, P. (2014). Semantic parsing via paraphrasing. In *ACL 2014*.
- Berg-Kirkpatrick, T.**, Burkett, D., and Klein, D. (2012). An empirical investigation of statistical significance in NLP. In *EMNLP 2012*, 995–1005.
- Berger, A.**, Della Pietra, S. A., and Della Pietra, V. J. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1), 39–71.
- Bergsma, S.** and Lin, D. (2006). Bootstrapping path-based pronoun resolution. In *COLING/ACL 2006*, 33–40.

- Bergsma**, S., Lin, D., and Goebel, R. (2008a). Discriminative learning of selectional preference from unlabeled text. In *EMNLP-08*, 59–68.
- Bergsma**, S., Lin, D., and Goebel, R. (2008b). Distributional identification of non-referential pronouns. In *ACL-08*, 10–18.
- Bethard**, S. (2013). ClearTK-TimeML: A minimalist approach to TempEval 2013. In *SemEval-13*, 10–14.
- Bhat**, I., Bhat, R. A., Shrivastava, M., and Sharma, D. (2017). Joining hands: Exploiting monolingual treebanks for parsing of code-mixing data. In *EACL-17*, 324–330.
- Biber**, D., Johansson, S., Leech, G., Conrad, S., and Finegan, E. (1999). *Longman Grammar of Spoken and Written English*. Pearson.
- Bies**, A., Ferguson, M., Katz, K., and MacIntyre, R. (1995). Bracketing guidelines for Treebank II style Penn Treebank Project..
- Bikel**, D. M., Miller, S., Schwartz, R., and Weischedel, R. (1997). Nymble: A high-performance learning name-finder. In *ANLP 1997*, 194–201.
- Biran**, O. and McKeown, K. (2015). PDTB discourse parsing as a tagging task: The two taggers approach. In *SIGDIAL 15*, 96–104.
- Bird**, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly.
- Bisani**, M. and Ney, H. (2004). Bootstrap estimates for confidence intervals in ASR performance evaluation. In *ICASSP-04*, Vol. I, 409–412.
- Bishop**, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- Bizer**, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., and Hellmann, S. (2009). DBpedia—A crystallization point for the Web of Data. *Web Semantics: science, services and agents on the world wide web*, 7(3), 154–165.
- Björkelund**, A. and Kuhn, J. (2014). Learning structured perceptrons for coreference resolution with latent antecedents and non-local features. In *ACL 2014*, 47–57.
- Black**, E. (1988). An experiment in computational discrimination of English word senses. *IBM Journal of Research and Development*, 32(2), 185–194.
- Black**, E., Abney, S. P., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J. L., Liberman, M. Y., Marcus, M. P., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings DARPA Speech and Natural Language Workshop*, 306–311.
- Black**, E., Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., and Roukos, S. (1992). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings DARPA Speech and Natural Language Workshop*, 134–139.
- Blei**, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *JMLR*, 3(5), 993–1022.
- Blodgett**, S. L., Green, L., and O'Connor, B. (2016). Demographic dialectal variation in social media: A case study of African-American English. In *EMNLP 2016*.
- Blodgett**, S. L. and O'Connor, B. (2017). Racial disparity in natural language processing: A case study of social media african-american english. In *Fairness, Accountability, and Transparency in Machine Learning (FAT/ML) Workshop, KDD*.
- Bloomfield**, L. (1914). *An Introduction to the Study of Language*. Henry Holt and Company.
- Bloomfield**, L. (1933). *Language*. University of Chicago Press.
- Bobrow**, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS: A frame driven dialog system. *Artificial Intelligence*, 8, 155–173.
- Bobrow**, D. G. and Norman, D. A. (1975). Some principles of memory schemata. In Bobrow, D. G. and Collins, A. (Eds.), *Representation and Understanding*. Academic Press.
- Bobrow**, D. G. and Winograd, T. (1977). An overview of KRL, a knowledge representation language. *Cognitive Science*, 1(1), 3–46.
- Bod**, R. (1993). Using an annotated corpus as a stochastic grammar. In *EACL-93*, 37–44.
- Boersma**, P. and Weenink, D. (2005). Praat: doing phonetics by computer (version 4.3.14). [Computer program]. Retrieved May 26, 2005, from <http://www.praat.org/>.
- Boguraev**, B. K. and Briscoe, T. (Eds.). (1989). *Computational Lexicography for Natural Language Processing*. Longman.
- Bohus**, D. and Rudnicky, A. I. (2005). Sorry, I didn't catch that! — An investigation of non-understanding errors and recovery strategies. In *Proceedings of SIGDIAL*.
- Bojanowski**, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *TACL*, 5, 135–146.
- Bolinger**, D. (1981). Two kinds of vowels, two kinds of rhythm. Indiana University Linguistics Club.
- Bollacker**, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD 2008*, 1247–1250.
- Bolukbasi**, T., Chang, K.-W., Zou, J., Saligrama, V., and Kalai, A. T. (2016). Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *NIPS 16*, 4349–4357.
- Booth**, T. L. (1969). Probabilistic representation of formal languages. In *IEEE Conference Record of the 1969 Tenth Annual Symposium on Switching and Automata Theory*, 74–81.
- Booth**, T. L. and Thompson, R. A. (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5), 442–450.
- Borges**, J. L. (1964). *The analytical language of John Wilkins*. University of Texas Press. Trans. Ruth L. C. Simms.
- Bowman**, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Jozefowicz, R., and Bengio, S. (2016). Generating sentences from a continuous space. In *CoNLL-16*, 10–21.
- Boyd-Graber**, J., Feng, S., and Rodriguez, P. (2018). Human-computer question answering: The case for quizbowl. In Escalera, S. and Weimer, M. (Eds.), *The NIPS '17 Competition: Building Intelligent Systems*. Springer.
- Brachman**, R. J. (1979). On the epistemological status of semantic networks. In Findler, N. V. (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, 3–50. Academic Press.
- Brachman**, R. J. and Levesque, H. J. (Eds.). (1985). *Readings in Knowledge Representation*. Morgan Kaufmann.
- Brachman**, R. J. and Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2), 171–216.
- Brants**, T. (2000). TnT: A statistical part-of-speech tagger. In *ANLP 2000*, 224–231.
- Brants**, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *EMNLP/CoNLL 2007*.
- Braud**, C., Coavoux, M., and Søgaard, A. (2017). Cross-lingual RST discourse parsing. In *EACL-17*, 292–304.

- Bréal, M.** (1897). *Essai de Sémantique: Science des significations*. Hachette.
- Brennan, S. E., Friedman, M. W., and Pollard, C.** (1987). A centering approach to pronouns. In *ACL-87*, 155–162.
- Bresnan, J. (Ed.)**. (1982). *The Mental Representation of Grammatical Relations*. MIT Press.
- Brill, E., Dumais, S. T., and Banko, M.** (2002). An analysis of the AskMSR question-answering system. In *EMNLP 2002*, 257–264.
- Brill, E. and Resnik, P.** (1994). A rule-based approach to prepositional phrase attachment disambiguation. In *COLING-94*, 1198–1204.
- Brin, S.** (1998). Extracting patterns and relations from the World Wide Web. In *Proceedings World Wide Web and Databases International Workshop, Number 1590 in LNCS*, 172–183. Springer.
- Briscoe, T. and Carroll, J.** (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1), 25–59.
- Brockmann, C. and Lapata, M.** (2003). Evaluating and combining approaches to selectional preference acquisition. In *EACL-03*, 27–34.
- Broschart, J.** (1997). Why Tongan does it differently. *Linguistic Typology*, 1, 123–165.
- Bruce, B. C.** (1975). Generation as a social action. In *Proceedings of TINLAP-1 (Theoretical Issues in Natural Language Processing)*, 64–67.
- Brysbaert, M., Warriner, A. B., and Kuperman, V.** (2014). Concreteness ratings for 40 thousand generally known English word lemmas. *Behavior Research Methods*, 46(3), 904–911.
- Buchholz, S. and Marsi, E.** (2006). CoNLL-x shared task on multilingual dependency parsing. In *CoNLL-06*, 149–164.
- Buck, C., Heafield, K., and Van Ooyen, B.** (2014). N-gram counts and language models from the common crawl. In *Proceedings of LREC*.
- Budanitsky, A. and Hirst, G.** (2006). Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1), 13–47.
- Budzianowski, P., Wen, T.-H., Tseng, B.-H., Casanueva, I., Ultes, S., Ramadan, O., and Gašić, M.** (2018). MultiWOZ - a large-scale multi-domain wizard-of-Oz dataset for task-oriented dialogue modelling. In *EMNLP 2018*, 5016–5026.
- Bullinaria, J. A. and Levy, J. P.** (2007). Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior research methods*, 39(3), 510–526.
- Bullinaria, J. A. and Levy, J. P.** (2012). Extracting semantic representations from word co-occurrence statistics: stop-lists, stemming, and SVD. *Behavior research methods*, 44(3), 890–907.
- Bulyko, I., Kirchhoff, K., Ostendorf, M., and Goldberg, J.** (2005). Error-sensitive response generation in a spoken language dialogue system. *Speech Communication*, 45(3), 271–288.
- Caliskan, A., Bryson, J. J., and Narayanan, A.** (2017). Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334), 183–186.
- Cardie, C.** (1993). A case-based approach to knowledge acquisition for domain specific sentence analysis. In *AAAI-93*, 798–803. AAAI Press.
- Cardie, C.** (1994). *Domain-Specific Knowledge Acquisition for Conceptual Sentence Analysis*. Ph.D. thesis, University of Massachusetts, Amherst, MA. Available as CMPSCI Technical Report 94-74.
- Cardie, C. and Wagstaff, K.** (1999). Noun phrase coreference as clustering. In *EMNLP/VLC-99*.
- Carlson, L. and Marcu, D.** (2001). Discourse tagging manual. Tech. rep. ISI-TR-545, ISI.
- Carlson, L., Marcu, D., and Okurowski, M. E.** (2001). Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of SIGDIAL*.
- Carpenter, R.** (2017). Cleverbot. <http://www.cleverbot.com>, accessed 2017.
- Carreras, X. and Márquez, L.** (2005). Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *CoNLL-05*, 152–164.
- Carroll, J., Briscoe, T., and Sanfilippo, A.** (1998). Parser evaluation: A survey and a new proposal. In *LREC-98*, 447–454.
- Chafe, W. L.** (1976). Givenness, contrastiveness, definiteness, subjects, topics, and point of view. In Li, C. N. (Ed.), *Subject and Topic*, 25–55. Academic Press.
- Chambers, N.** (2013). NavyTime: Event and time ordering from raw text. In *SemEval-13*, 73–77.
- Chambers, N. and Jurafsky, D.** (2010). Improving the use of pseudo-words for evaluating selectional preferences. In *ACL 2010*, 445–453.
- Chambers, N. and Jurafsky, D.** (2011). Template-based information extraction without the templates. In *ACL 2011*.
- Chang, A. X. and Manning, C. D.** (2012). SUTime: A library for recognizing and normalizing time expressions.. In *LREC-12*, 3735–3740.
- Chang, K.-W., Samdani, R., and Roth, D.** (2013). A constrained latent variable model for coreference resolution. In *EMNLP 2013*, 601–612.
- Chang, K.-W., Samdani, R., Rozovskaya, A., Sammons, M., and Roth, D.** (2012). Illinois-Coref: The UI system in the CoNLL-2012 shared task. In *CoNLL-12*, 113–117.
- Chaplot, D. S. and Salakhutdinov, R.** (2018). Knowledge-based word sense disambiguation using topic models. In *AAAI-18*.
- Charniak, E.** (1997). Statistical parsing with a context-free grammar and word statistics. In *AAAI-97*, 598–603.
- Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M.** (1993). Equations for part-of-speech tagging. In *AAAI-93*, 784–789. AAAI Press.
- Che, W., Li, Z., Li, Y., Guo, Y., Qin, B., and Liu, T.** (2009). Multilingual dependency-based syntactic and semantic parsing. In *CoNLL-09*, 49–54.
- Chelba, C. and Jelinek, F.** (2000). Structured language modeling. *Computer Speech and Language*, 14, 283–332.
- Chen, C. and Ng, V.** (2013). Linguistically aware coreference evaluation metrics. In *Sixth International Joint Conference on Natural Language Processing*, 1366–1374.
- Chen, D., Fisch, A., Weston, J., and Bordes, A.** (2017). Reading wikipedia to answer open-domain questions. In *ACL 2017*.
- Chen, D. and Manning, C. D.** (2014). A fast and accurate dependency parser using neural networks.. In *EMNLP 2014*, 740–750.
- Chen, E., Snyder, B., and Barzilay, R.** (2007). Incremental text structuring with online hierarchical ranking. In *EMNLP/CoNLL 2007*, 83–91.
- Chen, J. N. and Chang, J. S.** (1998). Topical clustering of MRD senses based on information retrieval techniques. *Computational Linguistics*, 24(1), 61–96.
- Chen, S. F. and Goodman, J.** (1996). An empirical study of smoothing techniques for language modeling. In *ACL-96*, 310–318.
- Chen, S. F. and Goodman, J.** (1998). An empirical study of smoothing techniques for language modeling. Tech. rep. TR-10-98, Computer Science Group, Harvard University.

- Chen, S. F.** and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13, 359–394.
- Chen, X.**, Shi, Z., Qiu, X., and Huang, X. (2017). Adversarial multi-criteria learning for Chinese word segmentation. In *ACL 2017*, 1193–1203.
- Chierchia, G.** and McConnell-Ginet, S. (1991). *Meaning and Grammar*. MIT Press.
- Chinchor, N.**, Hirschman, L., and Lewis, D. L. (1993). Evaluating Message Understanding systems: An analysis of the third Message Understanding Conference. *Computational Linguistics*, 19(3), 409–449.
- Chiticariu, L.**, Danilevsky, M., Li, Y., Reiss, F., and Zhu, H. (2018). SystemT: Declarative text understanding for enterprise. In *NAACL HLT 2018*, Vol. 3, 76–83.
- Chiticariu, L.**, Li, Y., and Reiss, F. R. (2013). Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems!. In *EMNLP 2013*, 827–832.
- Chiu, J. P. C.** and Nichols, E. (2016). Named entity recognition with bidirectional LSTM-CNNs. *TACL*, 4, 357–370.
- Cho, K.**, van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014*, 1724–1734.
- Choi, E.**, He, H., Iyyer, M., Yatskar, M., Yih, W.-t., Choi, Y., Liang, P., and Zettlemoyer, L. (2018). Quac: Question answering in context. In *EMNLP 2018*.
- Choi, J. D.** and Palmer, M. (2011a). Getting the most out of transition-based dependency parsing. In *ACL 2011*, 687–692.
- Choi, J. D.** and Palmer, M. (2011b). Transition-based semantic role labeling using predicate argument clustering. In *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*, 37–45.
- Choi, J. D.**, Tetreault, J., and Stent, A. (2015). It depends: Dependency parser comparison using a web-based evaluation tool. In *ACL 2015*, 26–31.
- Chomsky, N.** (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3), 113–124.
- Chomsky, N.** (1956/1975). *The Logical Structure of Linguistic Theory*. Plenum.
- Chomsky, N.** (1957). *Syntactic Structures*. Mouton, The Hague.
- Chomsky, N.** (1963). Formal properties of grammars. In Luce, R. D., Bush, R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. 2, 323–418. Wiley.
- Chomsky, N.** (1981). *Lectures on Government and Binding*. Foris.
- Christodoulopoulos, C.**, Goldwater, S., and Steedman, M. (2010). Two decades of unsupervised POS induction: How far have we come?. In *EMNLP-10*.
- Chu, Y.-J.** and Liu, T.-H. (1965). On the shortest arborescence of a directed graph. *Science Sinica*, 14, 1396–1400.
- Chu-Carroll, J.** (1998). A statistical model for discourse act recognition in dialogue interactions. In Chu-Carroll, J. and Green, N. (Eds.), *Applying Machine Learning to Discourse Processing. Papers from the 1998 AAAI Spring Symposium*. Tech. rep. SS-98-01, 12–17. AAAI Press.
- Chu-Carroll, J.** and Carberry, S. (1998). Collaborative response generation in planning dialogues. *Computational Linguistics*, 24(3), 355–400.
- Chu-Carroll, J.** and Carpenter, B. (1999). Vector-based natural language call routing. *Computational Linguistics*, 25(3), 361–388.
- Chu-Carroll, J.**, Fan, J., Boguraev, B. K., Carmel, D., Sheinwald, D., and Welty, C. (2012). Finding needles in the haystack: Search and candidate generation. *IBM Journal of Research and Development*, 56(3/4), 6:1–6:12.
- Church, A.** (1940). A formulation of a simple theory of types. *Journal of Symbolic Logic*, 5, 56–68.
- Church, K. W.** (1980). *On Memory Limitations in Natural Language Processing*. Master's thesis, MIT. Distributed by the Indiana University Linguistics Club.
- Church, K. W.** (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *ANLP 1988*, 136–143.
- Church, K. W.** (1989). A stochastic parts program and noun phrase parser for unrestricted text. In *ICASSP-89*, 695–698.
- Church, K. W.** (1994). Unix for Poets. Slides from 2nd ELSNET Summer School and unpublished paper ms.
- Church, K. W.** and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5, 19–54.
- Church, K. W.** and Hanks, P. (1989). Word association norms, mutual information, and lexicography. In *ACL 89*, 76–83.
- Church, K. W.** and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1), 22–29.
- Church, K. W.**, Hart, T., and Gao, J. (2007). Compressing trigram language models with Golomb coding. In *EMNLP/CoNLL 2007*, 199–207.
- Cialdini, R. B.** (1984). *Influence: The psychology of persuasion*. Morrow.
- Ciaramita, M.** and Altun, Y. (2006). Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *EMNLP 2006*, 594–602.
- Ciaramita, M.** and Johnson, M. (2003). Supersense tagging of unknown nouns in WordNet. In *EMNLP-2003*, 168–175.
- Clark, C.** and Gardner, M. (2018). Simple and effective multi-paragraph reading comprehension. In *ACL 2018*.
- Clark, E.** (1987). The principle of contrast: A constraint on language acquisition. In MacWhinney, B. (Ed.), *Mechanisms of language acquisition*, 1–33. LEA.
- Clark, H. H.** (1996). *Using Language*. Cambridge University Press.
- Clark, H. H.** and Fox Tree, J. E. (2002). Using uh and um in spontaneous speaking. *Cognition*, 84, 73–111.
- Clark, H. H.** and Marshall, C. (1981). Definite reference and mutual knowledge. In Joshi, A. K., Webber, B. L., and Sag, I. A. (Eds.), *Elements of Discourse Understanding*, 10–63. Cambridge.
- Clark, H. H.** and Wilkes-Gibbs, D. (1986). Referring as a collaborative process. *Cognition*, 22, 1–39.
- Clark, J.** and Yallop, C. (1995). *An Introduction to Phonetics and Phonology* (2nd Ed.). Blackwell.
- Clark, K.** and Manning, C. D. (2015). Entity-centric coreference resolution with model stacking. In *ACL 2015*, 1405–1415.
- Clark, K.** and Manning, C. D. (2016a). Deep reinforcement learning for mention-ranking coreference models. In *EMNLP 2016*.
- Clark, K.** and Manning, C. D. (2016b). Improving coreference resolution by learning entity-level distributed representations. In *ACL 2016*.
- Clark, P.**, Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? Try ARC, the AI2 reasoning challenge.. arXiv preprint arXiv:1803.05457.
- Clark, S.**, Curran, J. R., and Osborne, M. (2003). Bootstrapping pos taggers using unlabelled data. In *CoNLL-03*, 49–55.

- CMU** (1993). The Carnegie Mellon Pronouncing Dictionary v0.1. Carnegie Mellon University.
- Coccaro**, N. and Jurafsky, D. (1998). Towards better integration of semantic predictors in statistical language modeling. In *ICSLP-98*, 2403–2406.
- Cohen**, K. B. and Demner-Fushman, D. (2014). *Biomedical natural language processing*. Benjamins.
- Cohen**, M. H., Giangola, J. P., and Balogh, J. (2004). *Voice User Interface Design*. Addison-Wesley.
- Cohen**, P. R. and Perrault, C. R. (1979). Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3), 177–212.
- Colby**, K. M., Hilf, F. D., Weber, S., and Kraemer, H. C. (1972). Turing-like indistinguishability tests for the validation of a computer simulation of paranoid processes. *Artificial Intelligence*, 3, 199–221.
- Colby**, K. M., Weber, S., and Hilf, F. D. (1971). Artificial paranoia. *Artificial Intelligence*, 2(1), 1–25.
- Cole**, R. A., Novick, D. G., Vermeulen, P. J. E., Sutton, S., Fanty, M., Westsels, L. F. A., de Villiers, J. H., Schalkwyk, J., Hansen, B., and Burnett, D. (1997). Experiments with a spoken dialogue system for taking the US census. *Speech Communication*, 23, 243–260.
- Coleman**, J. (2005). *Introducing Speech and Language Processing*. Cambridge University Press.
- Collins**, M. (1996). A new statistical parser based on bigram lexical dependencies. In *ACL-96*, 184–191.
- Collins**, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.
- Collins**, M. (2003). Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4), 589–637.
- Collins**, M., Hajič, J., Ramshaw, L. A., and Tillmann, C. (1999). A statistical parser for Czech. In *ACL-99*, 505–512.
- Collobert**, R. and Weston, J. (2007). Fast semantic extraction using a novel neural network architecture. In *ACL-07*, 560–567.
- Collobert**, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 160–167.
- Collobert**, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *JMLR*, 12, 2493–2537.
- Connolly**, D., Burger, J. D., and Day, D. S. (1994). A machine learning approach to anaphoric reference. In *Proceedings of the International Conference on New Methods in Language Processing (NeMLaP)*.
- Copestake**, A. and Briscoe, T. (1995). Semi-productive polysemy and sense extension. *Journal of Semantics*, 12(1), 15–68.
- Cottrell**, G. W. (1985). *A Connectionist Approach to Word Sense Disambiguation*. Ph.D. thesis, University of Rochester, Rochester, NY. Revised version published by Pitman, 1989.
- Cover**, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley.
- Covington**, M. (2001). A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*, 95–102.
- Cox**, D. (1969). *Analysis of Binary Data*. Chapman and Hall, London.
- Craven**, M. and Kumlien, J. (1999). Constructing biological knowledge bases by extracting information from text sources. In *ISMB-99*, 77–86.
- Cruse**, D. A. (2004). *Meaning in Language: an Introduction to Semantics and Pragmatics*. Oxford University Press. Second edition.
- Cucerzan**, S. (2007). Large-scale named entity disambiguation based on Wikipedia data. In *EMNLP/CoNLL 2007*, 708–716.
- Culicover**, P. W. and Jackendoff, R. (2005). *Simpler Syntax*. Oxford University Press.
- Dagan**, I., Marcus, S., and Markovitch, S. (1993). Contextual word similarity and estimation from sparse data. In *ACL-93*, 164–171.
- Danieli**, M. and Gerbino, E. (1995). Metrics for evaluating dialogue strategies in a spoken language system. In *Proceedings of the 1995 AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, 34–39. AAAI Press.
- Das**, S. R. and Chen, M. Y. (2001). Yahoo! for Amazon: Sentiment parsing from small talk on the web. EFA 2001 Barcelona Meetings. Available at SSRN: <http://ssrn.com/abstract=276189>.
- Davidson**, D. (1967). The logical form of action sentences. In Rescher, N. (Ed.), *The Logic of Decision and Action*. University of Pittsburgh Press.
- Davidson**, T., Warmsley, D., Macy, M., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. In *ICWSM 2017*.
- Davies**, M. (2012). Expanding horizons in historical linguistics with the 400-million word Corpus of Historical American English. *Corpora*, 7(2), 121–157.
- Davies**, M. (2015). The Wikipedia Corpus: 4.6 million articles, 1.9 billion words. Adapted from Wikipedia. Available online at <https://www.english-corpora.org/wiki/>.
- Davis**, E. (1990). *Representations of Commonsense Knowledge*. Morgan Kaufmann.
- Davis**, E., Morgenstern, L., and Ortiz, C. L. (2017). The first Winograd schema challenge at IJCAI-16. *AI Magazine*, 38(3), 97–98.
- de Marneffe**, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., and Manning, C. D. (2014). Universal Stanford dependencies: A cross-linguistic typology.. In *LREC*, Vol. 14, 4585–92.
- de Marneffe**, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating typed dependency parses from phrase structure parses. In *LREC-06*.
- de Marneffe**, M.-C. and Manning, C. D. (2008). The stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, 1–8.
- de Marneffe**, M.-C., Recasens, M., and Potts, C. (2015). Modeling the lifespan of discourse entities with application to coreference resolution. *JAIR*, 52, 445–475.
- Deerwester**, S. C., Dumais, S. T., Furnas, G. W., Harshman, R. A., Landauer, T. K., Lochbaum, K. E., and Streeter, L. (1988). Computer information retrieval using latent semantic structure: US Patent 4,839,853..
- Deerwester**, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantics analysis. *JASIS*, 41(6), 391–407.
- DeJong**, G. F. (1982). An overview of the FRUMP system. In Lehnert, W. G. and Ringle, M. H. (Eds.), *Strategies for Natural Language Processing*, 149–176. LEA.
- Dempster**, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1), 1–21.
- Denis**, P. and Baldridge, J. (2007). Joint determination of anaphoricity and coreference resolution using integer programming. In *NAACL-HLT 07*.
- Denis**, P. and Baldridge, J. (2008). Specialized models and ranking for coreference resolution. In *EMNLP-08*, 660–669.

- Denis**, P. and Baldridge, J. (2009). Global joint models for coreference resolution and named entity classification. *Procesamiento del Lenguaje Natural*, 42.
- DeRose**, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14, 31–39.
- Devlin**, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019*, 4171–4186.
- Di Eugenio**, B. (1990). Centering theory and the Italian pronominal system. In *COLING-90*, 270–275.
- Di Eugenio**, B. (1996). The discourse functions of Italian subjects: A centering approach. In *COLING-96*, 352–357.
- Diab**, M. and Resnik, P. (2002). An unsupervised method for word sense tagging using parallel corpora. In *ACL-02*, 255–262.
- Digman**, J. M. (1990). Personality structure: Emergence of the five-factor model. *Annual Review of Psychology*, 41(1), 417–440.
- Ditman**, T. and Kuperberg, G. R. (2010). Building coherence: A framework for exploring the breakdown of links across clause boundaries in schizophrenia. *Journal of neurolinguistics*, 23(3), 254–269.
- Do**, Q. N. T., Bethard, S., and Moens, M.-F. (2017). Improving implicit semantic role labeling by predicting semantic frame arguments. In *IJCNLP-17*.
- Dolan**, B. (1994). Word sense ambiguation: Clustering related senses. In *COLING-94*, 712–716.
- dos Santos**, C. N., Xiang, B., and Zhou, B. (2015). Classifying relations by ranking with convolutional neural networks. In *ACL 2015*.
- Dowty**, D. R. (1979). *Word Meaning and Montague Grammar*. D. Reidel.
- Dowty**, D. R., Wall, R. E., and Peters, S. (1981). *Introduction to Montague Semantics*. D. Reidel.
- Dozat**, T., Qi, P., and Manning, C. D. (2017). Stanford’s graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task*, 20–30.
- Dror**, R., Baumer, G., Bogomolov, M., and Reichart, R. (2017). Replicability analysis for natural language processing: Testing significance with multiple datasets. *TACL*, 5, 471–486.
- Duda**, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Wiley and Sons.
- Durrett**, G. and Klein, D. (2013). Easy victories and uphill battles in coreference resolution. In *EMNLP 2013*.
- Durrett**, G. and Klein, D. (2014). A joint model for entity analysis: Coreference, typing, and linking. *TACL*, 2, 477–490.
- Earley**, J. (1968). *An Efficient Context-Free Parsing Algorithm*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.
- Earley**, J. (1970). An efficient context-free parsing algorithm. *CACM*, 6(8), 451–455.
- Edmonds**, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4), 233–240.
- Efron**, B. and Tibshirani, R. J. (1993). *An introduction to the bootstrap*. CRC press.
- Eggle**, L. (2007). Untangling Herdan’s law and Heaps’ law: Mathematical and informetric arguments. *JASIST*, 58(5), 702–709.
- Eisner**, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *COLING-96*, 340–345.
- Eisner**, J. (2002). An interactive spreadsheet for teaching the forward-backward algorithm. In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, 10–18.
- Ejerdé**, E. I. (1988). Finding clauses in unrestricted text by finitary and stochastic methods. In *ANLP 1988*, 219–227.
- Ekman**, P. (1999). Basic emotions. In Dalgleish, T. and Power, M. J. (Eds.), *Handbook of Cognition and Emotion*, 45–60. Wiley.
- Elman**, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2), 179–211.
- Elsner**, M., Austerweil, J., and Charniak, E. (2007). A unified local and global model for discourse coherence. In *NAACL-HLT 07*, 436–443.
- Elsner**, M. and Charniak, E. (2008). Coreference-inspired coherence modeling. In *ACL-08*, 41–44.
- Elsner**, M. and Charniak, E. (2011). Extending the entity grid with entity-specific features. In *ACL 2011*, 125–129.
- Elvevåg**, B., Foltz, P. W., Weinberger, D. R., and Goldberg, T. E. (2007). Quantifying incoherence in speech: an automated methodology and novel application to schizophrenia. *Schizophrenia research*, 93(1-3), 304–316.
- Emami**, A., Trichelair, P., Trischler, A., Suleiman, K., Schulz, H., and Cheung, J. C. K. (2019). The KNOWREF coreference corpus: Removing gender and number cues for difficult pronominal anaphora resolution. In *ACL 2019*.
- Erk**, K. (2007). A simple, similarity-based model for selectional preferences. In *ACL-07*, 216–223.
- Etzioni**, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. (2005). Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1), 91–134.
- Evans**, N. (2000). Word classes in the world’s languages. In Booij, G., Lehmann, C., and Mugdan, J. (Eds.), *Morphology: A Handbook on Inflection and Word Formation*, 708–732. Mouton.
- Fader**, A., Soderland, S., and Etzioni, O. (2011). Identifying relations for open information extraction. In *EMNLP-11*, 1535–1545.
- Fader**, A., Zettlemoyer, L., and Etzioni, O. (2013). Paraphrase-driven learning for open question answering. In *ACL 2013*, 1608–1618.
- Fano**, R. M. (1961). *Transmission of Information: A Statistical Theory of Communications*. MIT Press.
- Fant**, G. M. (1960). *Acoustic Theory of Speech Production*. Mouton.
- Fant**, G. M. (2004). *Speech Acoustics and Phonetics*. Kluwer.
- Faruqui**, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (2015). Retrofitting word vectors to semantic lexicons. In *NAACL HLT 2015*, 1606–1615.
- Fast**, E., Chen, B., and Bernstein, M. S. (2016). Empath: Understanding Topic Signals in Large-Scale Text. In *CHI*.
- Fauconnier**, G. and Turner, M. (2008). *The way we think: Conceptual blending and the mind’s hidden complexities*. Basic Books.
- Fazel-Zarandi**, M., Li, S.-W., Cao, J., Casale, J., Henderson, P., Whitney, D., and Geramifard, A. (2017). Learning robust dialog policies in noisy environments. In *Conversational AI Workshop (NIPS)*.
- Feldman**, J. A. and Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science*, 6, 205–254.
- Fellbaum**, C. (Ed.). (1998). *WordNet: An Electronic Lexical Database*. MIT Press.
- Feng**, V. W. and Hirst, G. (2011). Classifying arguments by scheme. In *ACL 2011*, 987–996.

- Feng**, V. W. and Hirst, G. (2014). A linear-time bottom-up discourse parser with constraints and post-editing. In *ACL 2014*, 511–521.
- Feng**, V. W., Lin, Z., and Hirst, G. (2014). The impact of deep hierarchical discourse structures in the evaluation of text coherence. In *COLING-14*, 940–949.
- Fensel**, D., Hendler, J. A., Lieberman, H., and Wahlster, W. (Eds.). (2003). *Spinning the Semantic Web: Bring the World Wide Web to its Full Potential*. MIT Press, Cambridge, MA.
- Fernandes**, E. R., dos Santos, C. N., and Milićić, R. L. (2012). Latent structure perceptron with feature induction for unrestricted coreference resolution. In *CoNLL-12*, 41–48.
- Ferro**, L., Gerber, L., Mani, I., Sundheim, B., and Wilson, G. (2005). Tides 2005 standard for the annotation of temporal expressions. Tech. rep., MITRE.
- Ferrucci**, D. A. (2012). Introduction to “This is Watson”. *IBM Journal of Research and Development*, 56(3/4), 1:1–1:15.
- Fessler**, L. (2017). We tested bots like Siri and Alexa to see who would stand up to sexual harassment. In *Quartz*. Feb 22, 2017. <https://qz.com/911681/>.
- Field**, A. and Tsvetkov, Y. (2019). Entity-centric contextual affective analysis. In *ACL 2019*, 2550–2560.
- Fikes**, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189–208.
- Fillmore**, C. J. (1966). A proposal concerning English prepositions. In Dinneen, F. P. (Ed.), *17th annual Round Table*, Vol. 17 of *Monograph Series on Language and Linguistics*, 19–34. Georgetown University Press.
- Fillmore**, C. J. (1968). The case for case. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, 1–88. Holt, Rinehart & Winston.
- Fillmore**, C. J. (1985). Frames and the semantics of understanding. *Quaderni di Semantica*, VI(2), 222–254.
- Fillmore**, C. J. (2003). Valency and semantic roles: the concept of deep structure case. In Ágel, V., Eichinger, L. M., Eroms, H. W., Hellwig, P., Heringer, H. J., and Lobin, H. (Eds.), *Dependenz und Valenz: Ein internationales Handbuch der zeitgenössischen Forschung*, chap. 36, 457–475. Walter de Gruyter.
- Fillmore**, C. J. (2012). Encounters with language. *Computational Linguistics*, 38(4), 701–718.
- Fillmore**, C. J. and Baker, C. F. (2009). A frames approach to semantic analysis. In Heine, B. and Narrog, H. (Eds.), *The Oxford Handbook of Linguistic Analysis*, 313–340. Oxford University Press.
- Fillmore**, C. J., Johnson, C. R., and Petrucc, M. R. L. (2003). Background to FrameNet. *International journal of lexicography*, 16(3), 235–250.
- Finkelstein**, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfson, G., and Ruppin, E. (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, 20(1), 116–131.
- Finlayson**, M. A. (2016). Inferring propp’s functions from semantically annotated text. *The Journal of American Folklore*, 129(511), 55–77.
- Firth**, J. R. (1935). The technique of semantics. *Transactions of the philological society*, 34(1), 36–73.
- Firth**, J. R. (1957). A synopsis of linguistic theory 1930–1955. In *Studies in Linguistic Analysis*. Philological Society. Reprinted in Palmer, F. (ed.) 1968. Selected Papers of J. R. Firth. Longman, Harlow.
- Fitt**, S. (2002). Unisyn lexicon. <http://www.cstr.ed.ac.uk/projects/unisyn/>.
- Poland**, W. and Martin, J. H. (2016). CU-NLP at semeval-2016 task 8: AMR parsing using LSTM-based recurrent neural networks. In *Proceedings of the 10th International Workshop on Semantic Evaluation*, 1197–1201.
- Poland**, Jr., W. R. and Martin, J. H. (2015). Dependency-based semantic role labeling using convolutional neural networks. In **SEM 2015*, 279–289.
- Foltz**, P. W., Kintsch, W., and Landauer, T. K. (1998). The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2–3), 285–307.
- Forchini**, P. (2013). Using movie corpora to explore spoken American English: Evidence from multidimensional analysis. In Bamford, J., Cavalieri, S., and Diani, G. (Eds.), *Variation and Change in Spoken and Written Discourse: Perspectives from corpus linguistics*, 123–136. Benjamins.
- Forney, Jr.**, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3), 268–278.
- Fox**, B. A. (1993). *Discourse Structure and Anaphora: Written and Conversational English*. Cambridge.
- Francis**, H. S., Gregory, M. L., and Michaelis, L. A. (1999). Are lexical subjects deviant? In *CLS-99*. University of Chicago.
- Francis**, W. N. and Kučera, H. (1982). *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.
- Franz**, A. (1997). Independence assumptions considered harmful. In *ACL/EACL-97*, 182–189.
- Franz**, A. and Brants, T. (2006). All our n-gram are belong to you. <http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html>.
- Fraser**, N. M. and Gilbert, G. N. (1991). Simulating speech systems. *Computer Speech and Language*, 5, 81–99.
- Fry**, D. B. (1955). Duration and intensity as physical correlates of linguistic stress. *JASA*, 27, 765–768.
- Fyshe**, A., Wehbe, L., Talukdar, P. P., Murphy, B., and Mitchell, T. M. (2015). A compositional and interpretable semantic space. In *NAACL HLT 2015*.
- Gabow**, H. N., Galil, Z., Spencer, T., and Tarjan, R. E. (1986). Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2), 109–122.
- Gage**, P. (1994). A new algorithm for data compression. *The C Users Journal*, 12(2), 23–38.
- Gale**, W. A. and Church, K. W. (1994). What is wrong with adding one?. In Oostdijk, N. and de Haan, P. (Eds.), *Corpus-Based Research into Language*, 189–198. Rodopi.
- Gale**, W. A., Church, K. W., and Yarowsky, D. (1992a). Estimating upper and lower bounds on the performance of word-sense disambiguation programs. In *ACL-92*, 249–256.
- Gale**, W. A., Church, K. W., and Yarowsky, D. (1992b). One sense per discourse. In *Proceedings DARPA Speech and Natural Language Workshop*, 233–237.
- Gale**, W. A., Church, K. W., and Yarowsky, D. (1992c). Work on statistical methods for word sense disambiguation. In Goldman, R. (Ed.), *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.
- Gao**, S., Sethi, A., Aggarwal, S., Chung, T., and Hakkani-Tür, D. (2019). Dialog state tracking: A neural reading comprehension approach. arXiv preprint arXiv:1908.01946.
- Garg**, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16), E3635–E3644.

- Garside, R.** (1987). The CLAWS word-tagging system. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, 30–41. Longman.
- Garside, R.**, Leech, G., and McEnergy, A. (1997). *Corpus Annotation*. Longman.
- Gazdar, G.**, Klein, E., Pullum, G. K., and Sag, I. A. (1985). *Generalized Phrase Structure Grammar*. Blackwell.
- Gerber, M.** and Chai, J. Y. (2010). Beyond nombank: A study of implicit arguments for nominal predicates. In *ACL 2010*, 1583–1592.
- Gil, D.** (2000). Syntactic categories, cross-linguistic variation and universal grammar. In Vogel, P. M. and Comrie, B. (Eds.), *Approaches to the Typology of Word Classes*, 173–216. Mouton.
- Gildea, D.** and Jurafsky, D. (2000). Automatic labeling of semantic roles. In *ACL-00*, 512–520.
- Gildea, D.** and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, 28(3), 245–288.
- Gildea, D.** and Palmer, M. (2002). The necessity of syntactic parsing for predicate argument recognition. In *ACL-02*.
- Giles, C. L.**, Kuhn, G. M., and Williams, R. J. (1994). Dynamic recurrent neural networks: Theory and applications. *IEEE Trans. Neural Netw. Learning Syst.*, 5(2), 153–156.
- Gillick, L.** and Cox, S. J. (1989). Some statistical issues in the comparison of speech recognition algorithms. In *ICASSP-89*, 532–535.
- Ginzburg, J.** and Sag, I. A. (2000). *Interrogative Investigations: the Form, Meaning and Use of English Interrogatives*. CSLI.
- Girard, G.** (1718). *La justesse de la langue françoise: ou les différentes significations des mots qui passent pour synonymes*.
- Giuliano, V. E.** (1965). The interpretation of word associations. In Stevens, M. E., Giuliano, V. E., and Heilprin, L. B. (Eds.), *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*, 25–32. <https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscriminatory.pdf>.
- Givón, T.** (1990). *Syntax: A Functional Typological Introduction*. John Benjamins.
- Glennie, A.** (1960). On the syntax machine and the construction of a universal compiler. Tech. rep. No. 2, Contr. NR 049-141, Carnegie Mellon University (at the time Carnegie Institute of Technology), Pittsburgh, PA.
- Godfrey, J.**, Holliman, E., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *ICASSP-92*, 517–520.
- Goffman, E.** (1974). *Frame analysis: An essay on the organization of experience*. Harvard University Press.
- Gold, B.** and Morgan, N. (1999). *Speech and Audio Signal Processing*. Wiley Press.
- Goldberg, J.**, Ostendorf, M., and Kirchhoff, K. (2003). The impact of response wording in error correction subdialogs. In *ISCA Tutorial and Research Workshop on Error Handling in Spoken Dialogue Systems*.
- Goldberg, Y.** (2017). *Neural Network Methods for Natural Language Processing*, Vol. 10 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool.
- Gondek, D. C.**, Lally, A., Kalyanpur, A., Murdock, J. W., Duboué, P. A., Zhang, L., Pan, Y., Qiu, Z. M., and Welty, C. (2012). A framework for merging and ranking of answers in deepqa. *IBM Journal of Research and Development*, 56(3/4), 14:1–14:12.
- Gonen, H.** and Goldberg, Y. (2019). Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. In *NAACL HLT 2019*.
- Good, M. D.**, Whiteside, J. A., Wixon, D. R., and Jones, S. J. (1984). Building a user-derived interface. *CACM*, 27(10), 1032–1043.
- Goodfellow, I.**, Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- Goodman, J.** (2006). A bit of progress in language modeling: Extended version. Tech. rep. MSR-TR-2001-72, Machine Learning and Applied Statistics Group, Microsoft Research, Redmond, WA.
- Goodwin, C.** (1996). Transparent vision. In Ochs, E., Schegloff, E. A., and Thompson, S. A. (Eds.), *Interaction and Grammar*, 370–404. Cambridge University Press.
- Gopalakrishnan, K.**, Hedayatnia, B., Chen, Q., Gottardi, A., Kwatra, S., Venkatesh, A., Gabriel, R., and Hakkani-Tür, D. (2019). Topical-chat: Towards knowledge-grounded open-domain conversations..
- Gould, J. D.**, Conti, J., and Hovanyecz, T. (1983). Composing letters with a simulated listening typewriter. *CACM*, 26(4), 295–308.
- Gould, J. D.** and Lewis, C. (1985). Designing for usability: Key principles and what designers think. *CACM*, 28(3), 300–311.
- Gould, S. J.** (1980). *The Panda's Thumb*. Penguin Group.
- Gravano, A.**, Hirschberg, J., and Beňuš, Š. (2012). Affirmative cue words in task-oriented dialogue. *Computational Linguistics*, 38(1), 1–39.
- Graves, A.** (2013). Generating sequences with recurrent neural networks.
- Graves, A.**, Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, 369–376.
- Graves, A.**, Fernández, S., Liwicki, M., Bunke, H., and Schmidhuber, J. (2007). Unconstrained on-line handwriting recognition with recurrent neural networks. In *NIPS 2007*, 577–584.
- Graves, A.**, Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 6645–6649.
- Graves, A.** and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6), 602–610.
- Green, B. F.**, Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question answerer. In *Proceedings of the Western Joint Computer Conference 19*, 219–224. Reprinted in Grosz et al. (1986).
- Greenberg, S.**, Ellis, D., and Hollenbach, J. (1996). Insights into spoken language gleaned from phonetic transcription of the Switchboard corpus. In *ICSLP-96*, S24–27.
- Greene, B. B.** and Rubin, G. M. (1971). Automatic grammatical tagging of English. Department of Linguistics, Brown University, Providence, Rhode Island.
- Greenwald, A. G.**, McGhee, D. E., and Schwartz, J. L. K. (1998). Measuring individual differences in implicit cognition: the implicit association test. *Journal of personality and social psychology*, 74(6), 1464–1480.
- Grenager, T.** and Manning, C. D. (2006). Unsupervised Discovery of a Statistical Verb Lexicon. In *EMNLP 2006*.
- Grice, H. P.** (1975). Logic and conversation. In Cole, P. and Morgan, J. L.

- (Eds.), *Speech Acts: Syntax and Semantics Volume 3*, 41–58. Academic Press.
- Grice, H. P.** (1978). Further notes on logic and conversation. In Cole, P. (Ed.), *Pragmatics: Syntax and Semantics Volume 9*, 113–127. Academic Press.
- Grishman, R.** and Sundheim, B. (1995). Design of the MUC-6 evaluation. In *MUC-6*, 1–11.
- Grosz, B. J.** (1977a). The representation and use of focus in a system for understanding dialogs. In *IJCAI-77*, 67–76. Morgan Kaufmann. Reprinted in Grosz et al. (1986).
- Grosz, B. J.** (1977b). *The Representation and Use of Focus in Dialogue Understanding*. Ph.D. thesis, University of California, Berkeley.
- Grosz, B. J., Joshi, A. K., and Weinstein, S.** (1983). Providing a unified account of definite noun phrases in English. In *ACL-83*, 44–50.
- Grosz, B. J., Joshi, A. K., and Weinstein, S.** (1995). Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2), 203–225.
- Grosz, B. J. and Sidner, C. L.** (1980). Plans for discourse. In Cohen, P. R., Morgan, J., and Pollack, M. E. (Eds.), *Intentions in Communication*, 417–444. MIT Press.
- Gruber, J. S.** (1965). *Studies in Lexical Relations*. Ph.D. thesis, MIT.
- Guinaudeau, C. and Strube, M.** (2013). Graph-based local coherence modeling. In *ACL 2013*, 93–103.
- Guindon, R.** (1988). A multidisciplinary perspective on dialogue structure in user-advisor dialogues. In Guindon, R. (Ed.), *Cognitive Science and Its Applications for Human-Computer Interaction*, 163–200. Lawrence Erlbaum.
- Gundel, J. K., Hedberg, N., and Zacharski, R.** (1993). Cognitive status and the form of referring expressions in discourse. *Language*, 69(2), 274–307.
- Gusfield, D.** (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.
- Guyon, I. and Elisseeff, A.** (2003). An introduction to variable and feature selection. *JMLR*, 3, 1157–1182.
- Habernal, I. and Gurevych, I.** (2016). Which argument is more convincing? Analyzing and predicting convincingness of Web arguments using bidirectional LSTM. In *ACL 2016*, 1589–1599.
- Habernal, I. and Gurevych, I.** (2017). Argumentation mining in user-generated web discourse. *Computational Linguistics*, 43(1), 125–179.
- Haghghi, A. and Klein, D.** (2009). Simple coreference resolution with rich syntactic and semantic features. In *EMNLP-09*, 1152–1161.
- Hajishirzi, H., Zilles, L., Weld, D. S., and Zettlemoyer, L.** (2013). Joint coreference resolution and named-entity linking with multi-pass sieves. In *EMNLP 2013*, 289–299.
- Hajic̄, J.** (1998). *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank*, 106–132. Karolinum.
- Hajic̄, J.** (2000). Morphological tagging: Data vs. dictionaries. In *NAACL 2000*. Seattle.
- Hajic̄, J., Caramita, M., Johansson, R., Kawahara, D., Martí, M. A., Márquez, L., Meyers, A., Nirve, J., Padó, S., Štěpánek, J., Stranák, P., Surdeanu, M., Xue, N., and Zhang, Y.** (2009). The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *CoNLL-09*, 1–18.
- Hakkani-Tür, D., Oflazer, K., and Tür, G.** (2002). Statistical morphological disambiguation for agglutinative languages. *Journal of Computers and Humanities*, 36(4), 381–410.
- Halliday, M. A. K. and Hasan, R.** (1976). *Cohesion in English*. Longman. English Language Series, Title No. 9.
- Hamilton, W. L., Clark, K., Leskovec, J., and Jurafsky, D.** (2016a). Inducing domain-specific sentiment lexicons from unlabeled corpora. In *EMNLP 2016*.
- Hamilton, W. L., Leskovec, J., and Jurafsky, D.** (2016b). Diachronic word embeddings reveal statistical laws of semantic change. In *ACL 2016*.
- Harabagiu, S., Pasca, M., and Maiorano, S.** (2000). Experiments with open-domain textual question answering. In *COLING-00*.
- Harris, R. A.** (2005). *Voice Interaction Design: Crafting the New Conversational Speech Systems*. Morgan Kaufmann.
- Harris, Z. S.** (1946). From morpheme to utterance. *Language*, 22(3), 161–183.
- Harris, Z. S.** (1954). Distributional structure. *Word*, 10, 146–162. Reprinted in J. Fodor and J. Katz, *The Structure of Language*, Prentice Hall, 1964 and in Z. S. Harris, *Papers in Structural and Transformational Linguistics*, Reidel, 1970, 775–794.
- Harris, Z. S.** (1962). *String Analysis of Sentence Structure*. Mouton, The Hague.
- Hastie, T., Tibshirani, R. J., and Friedman, J. H.** (2001). *The Elements of Statistical Learning*. Springer.
- Hatzivassiloglou, V. and McKeown, K.** (1997). Predicting the semantic orientation of adjectives. In *ACL/EACL-97*, 174–181.
- Hatzivassiloglou, V. and Wiebe, J.** (2000). Effects of adjective orientation and gradability on sentence subjectivity. In *COLING-00*, 299–305.
- Haviland, S. E. and Clark, H. H.** (1974). What's new? Acquiring new information as a process in comprehension. *Journal of Verbal Learning and Verbal Behaviour*, 13, 512–521.
- Hawkins, J. A.** (1978). *Definiteness and indefiniteness: a study in reference and grammaticality prediction*. Croom Helm Ltd.
- He, L., Lee, K., Lewis, M., and Zettlemoyer, L.** (2017). Deep semantic role labeling: What works and what's next. In *ACL 2017*, 473–483.
- Heafield, K.** (2011). KenLM: Faster and smaller language model queries. In *Workshop on Statistical Machine Translation*, 187–197.
- Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P.** (2013). Scalable modified Kneser-Ney language model estimation.. In *ACL 2013*, 690–696.
- Heaps, H. S.** (1978). *Information retrieval. Computational and theoretical aspects*. Academic Press.
- Hearst, M. A.** (1991). Noun homograph disambiguation. In *Proceedings of the 7th Conference of the University of Waterloo Centre for the New OED and Text Research*, 1–19.
- Hearst, M. A.** (1992a). Automatic acquisition of hyponyms from large text corpora. In *COLING-92*.
- Hearst, M. A.** (1992b). Automatic acquisition of hyponyms from large text corpora. In *COLING-92*. COLING.
- Hearst, M. A.** (1997). Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational Linguistics*, 23, 33–64.
- Hearst, M. A.** (1998). Automatic discovery of WordNet relations. In Fellbaum, C. (Ed.), *WordNet: An Electronic Lexical Database*. MIT Press.
- Heckerman, D., Horvitz, E., Sahami, M., and Dumais, S. T.** (1998). A bayesian approach to filtering junk e-mail. In *Proceeding of AAAI-98 Workshop on Learning for Text Categorization*, 55–62.
- Heim, I.** (1982). *The semantics of definite and indefinite noun phrases*. Ph.D. thesis, University of Massachusetts at Amherst.
- Heim, I. and Kratzer, A.** (1998). *Semantics in a Generative Grammar*. Blackwell Publishers, Malden, MA.

- Heinz, J. M.** and Stevens, K. N. (1961). On the properties of voiceless fricative consonants. *JASA*, 33, 589–596.
- Hellrich, J.**, Buechel, S., and Hahn, U. (2019). Modeling word emotion in historical language: Quantity beats supposed stability in seed word selection. In *Proceedings of the 3rd Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature*, 1–11.
- Hemphill, C. T.**, Godfrey, J., and Dodgington, G. (1990). The ATIS spoken language systems pilot corpus. In *Proceedings DARPA Speech and Natural Language Workshop*, 96–101.
- Henderson, P.**, Sinha, K., Angelard-Gontier, N., Ke, N. R., Fried, G., Lowe, R., and Pineau, J. (2017). Ethical challenges in data-driven dialogue systems. In *AAAI/ACM AI Ethics and Society Conference*.
- Hendrickx, I.**, Kim, S. N., Kozareva, Z., Nakov, P., Ó Séaghdha, D., Padó, S., Pennacchiotti, M., Romano, L., and Szpakowicz, S. (2009). Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, 94–99.
- Hendrix, G. G.**, Thompson, C. W., and Slocum, J. (1973). Language processing via canonical verbs and semantic models. In *Proceedings of IJCAI-73*.
- Henrich, V.**, Hinrichs, E., and Vodolazova, T. (2012). WebCAGe – a web-harvested corpus annotated with Germanet senses. In *EACL-12*, 387–396.
- Herdan, G.** (1960). *Type-token mathematics*. The Hague, Mouton.
- Hermann, K. M.**, Kociiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, 1693–1701.
- Hernault, H.**, Prendinger, H., duVerle, D. A., and Ishizuka, M. (2010). Hilda: A discourse parser using support vector machine classification. *Dialogue & Discourse*, 1(3).
- Hidey, C.**, Musi, E., Hwang, A., Mureşan, S., and McKeown, K. (2017). Analyzing the semantic types of claims and premises in an online persuasive forum. In *Proceedings of the 4th Workshop on Argument Mining*, 11–21.
- Hill, F.**, Reichart, R., and Korhonen, A. (2015). Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4), 665–695.
- Hindle, D.** and Rooth, M. (1990). Structural ambiguity and lexical relations. In *Proceedings DARPA Speech and Natural Language Workshop*, 257–262.
- Hindle, D.** and Rooth, M. (1991). Structural ambiguity and lexical relations. In *ACL-91*, 229–236.
- Hinkelmann, E. A.** and Allen, J. (1989). Two constraints on speech act ambiguity. In *ACL-89*, 212–219.
- Hinton, G. E.** (1986). Learning distributed representations of concepts. In *COGSCI-86*, 1–12.
- Hinton, G. E.**, Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7), 1527–1554.
- Hinton, G. E.**, Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.
- Hirschberg, J.**, Litman, D. J., and Swerts, M. (2001). Identifying user corrections automatically in spoken dialogue systems. In *NAACL 2001*.
- Hirschberg, J.** and Pierrehumbert, J. B. (1986). The intonational structuring of discourse. In *ACL-86*, 136–144.
- Hirschman, L.**, Light, M., Breck, E., and Burger, J. D. (1999). Deep Read: A reading comprehension system. In *ACL-99*, 325–332.
- Hirschman, L.** and Pao, C. (1993). The cost of errors in a spoken language system. In *EUROSPEECH-93*, 1419–1422.
- Hirst, G.** (1981). *Anaphora in Natural Language Understanding: A survey*. No. 119 in Lecture notes in computer science. Springer-Verlag.
- Hirst, G.** (1987). *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge University Press.
- Hirst, G.** (1988). Resolving lexical ambiguity computationally with spreading activation and polaroid words. In Small, S. L., Cottrell, G. W., and Tanenhaus, M. K. (Eds.), *Lexical Ambiguity Resolution*, 73–108. Morgan Kaufmann.
- Hirst, G.** and Charniak, E. (1982). Word sense and case slot disambiguation. In *AAAI-82*, 95–98.
- Hjelmslev, L.** (1969). *Prologomena to a Theory of Language*. University of Wisconsin Press. Translated by Francis J. Whitfield; original Danish edition 1943.
- Hobbs, J. R.** (1978). Resolving pronoun references. *Lingua*, 44, 311–338. Reprinted in Grosz et al. (1986).
- Hobbs, J. R.** (1979). Coherence and coreference. *Cognitive Science*, 3, 67–90.
- Hobbs, J. R.**, Appelt, D. E., Bear, J., Israel, D., Kamayama, M., Stickel, M. E., and Tyson, M. (1997). FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In Roche, E. and Schabes, Y. (Eds.), *Finite-State Language Processing*, 383–406. MIT Press.
- Hochreiter, S.** and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
- Hockenmaier, J.** and Steedman, M. (2007). Ccgbank: a corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3), 355–396.
- Hofmann, T.** (1999). Probabilistic latent semantic indexing. In *SIGIR-99*.
- Hopcroft, J. E.** and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.
- Hou, Y.**, Markert, K., and Strube, M. (2018). Unrestricted bridging resolution. *Computational Linguistics*, 44(2), 237–284.
- Householder, F. W.** (1995). Dionysius Thrax, the *technai*, and Sextus Empiricus. In Koerner, E. F. K. and Asher, R. E. (Eds.), *Concise History of the Language Sciences*, 99–103. Elsevier Science.
- Hovy, E. H.** (1990). Parsimonious and profligate approaches to the question of discourse structure relations. In *Proceedings of the 5th International Workshop on Natural Language Generation*, 128–136.
- Hovy, E. H.**, Hermjakob, U., and Ravichandran, D. (2002). A question/answer typology with surface text patterns. In *HLT-01*.
- Hovy, E. H.**, Marcus, M. P., Palmer, M., Ramshaw, L. A., and Weischedel, R. (2006). Ontonotes: The 90% solution. In *HLT-NAACL-06*.
- Hu, M.** and Liu, B. (2004a). Mining and summarizing customer reviews. In *KDD*, 168–177.
- Hu, M.** and Liu, B. (2004b). Mining and summarizing customer reviews. In *SIGKDD*.
- Huang, E. H.**, Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *ACL 2012*, 873–882.
- Huang, L.** and Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *ACL 2010*, 1077–1086.
- Huang, X.**, Acero, A., and Hon, H.-W. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall.

- Huang, Z., Xu, W., and Yu, K.** (2015). Bidirectional LSTM-CRF models for sequence tagging. In *arXiv preprint arXiv:1508.01991*.
- Huddleston, R.** and **Pullum, G. K.** (2002). *The Cambridge Grammar of the English Language*. Cambridge University Press.
- Hudson, R. A.** (1984). *Word Grammar*. Blackwell.
- Huffman, S.** (1996). Learning information extraction patterns from examples. In Wertmer, S., Riloff, E., and Scheller, G. (Eds.), *Connectionist, Statistical, and Symbolic Approaches to Learning Natural Language Processing*, 246–260. Springer.
- Hutto, C. J., Folds, D., and Appling, S.** (2015). Computationally detecting and quantifying the degree of bias in sentence-level text of news stories. In *HUSO 2015: The First International Conference on Human and Social Analytics*.
- Hymes, D.** (1974). Ways of speaking. In Bauman, R. and Sherzer, J. (Eds.), *Explorations in the ethnography of speaking*, 433–451. Cambridge University Press.
- Iacobacci, I., Pilehvar, M. T., and Navigli, R.** (2016). Embeddings for word sense disambiguation: An evaluation study. In *ACL 2016*, 897–907.
- Iida, R., Inui, K., Takamura, H., and Matsumoto, Y.** (2003). Incorporating contextual cues in trainable models for coreference resolution. In *EACL Workshop on The Computational Treatment of Anaphora*.
- Irons, E. T.** (1961). A syntax directed compiler for ALGOL 60. *CACM*, 4, 51–55.
- Irsay, O.** and **Cardie, C.** (2014). Opinion mining with deep recurrent neural networks. In *EMNLP 2014*, 720–728.
- Isbell, C. L., Kearns, M., Kormann, D., Singh, S., and Stone, P.** (2000). Cobot in LambdaMOO: A social statistics agent. In *AAAI/IAAI*, 36–41.
- ISO8601** (2004). Data elements and interchange formats—information interchange—representation of dates and times. Tech. rep., International Organization for Standards (ISO).
- Jackendoff, R.** (1983). *Semantics and Cognition*. MIT Press.
- Jacobs, P. S.** and **Rau, L. F.** (1990). SCISOR: A system for extracting information from on-line news. *CACM*, 33(11), 88–97.
- Jaech, A., Mulcaire, G., Hathi, S., Ostendorf, M., and Smith, N. A.** (2016). Hierarchical character-word models for language identification. In *ACL Workshop on NLP for Social Media*, 84–93.
- Jafarpour, S., Burges, C. J. C., and Ritter, A.** (2009). Filter, rank, and transfer the knowledge: Learning to chat. In *NIPS Workshop on Advances in Ranking*.
- Jauhainen, T., Lui, M., Zampieri, M., Baldwin, T., and Lindén, K.** (2018). Automatic language identification in texts: A survey. *arXiv preprint arXiv:1804.08186*.
- Jefferson, G.** (1972). Side sequences. In Sudnow, D. (Ed.), *Studies in social interaction*, 294–333. Free Press, New York.
- Jeffreys, H.** (1948). *Theory of Probability* (2nd Ed.). Clarendon Press. Section 3.23.
- Jelinek, F.** (1976). Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4), 532–557.
- Jelinek, F.** (1990). Self-organized language modeling for speech recognition. In Waibel, A. and Lee, K.-F. (Eds.), *Readings in Speech Recognition*, 450–506. Morgan Kaufmann. Originally distributed as IBM technical report in 1985.
- Jelinek, F.** (1997). *Statistical Methods for Speech Recognition*. MIT Press.
- Jelinek, F.** and **Lafferty, J. D.** (1991). Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3), 315–323.
- Jelinek, F., Lafferty, J. D., Magerman, D. M., Mercer, R. L., Ratnaparkhi, A., and Roukos, S.** (1994). Decision tree parsing using a hidden derivation model. In *ARPA Human Language Technologies Workshop*, 272–277.
- Jelinek, F.** and **Mercer, R. L.** (1980). Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. N. (Eds.), *Proceedings, Workshop on Pattern Recognition in Practice*, 381–397. North Holland.
- Ji, H.** and **Grishman, R.** (2011). Knowledge base population: Successful approaches and challenges. In *ACL 2011*, 1148–1158.
- Ji, H., Grishman, R., and Dang, H. T.** (2010). Overview of the tac 2011 knowledge base population track. In *TAC-11*.
- Ji, Y.** and **Eisenstein, J.** (2014). Representation learning for text-level discourse parsing. In *ACL 2014*, 13–24.
- Ji, Y.** and **Eisenstein, J.** (2015). One vector is not enough: Entity-augmented distributed semantics for discourse relations. *TACL*, 3, 329–344.
- Johnson, K.** (2003). *Acoustic and Auditory Phonetics* (2nd Ed.). Blackwell.
- Johnson, M.** (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4), 613–632.
- Johnson, W. E.** (1932). Probability: deductive and inductive problems (appendix to). *Mind*, 41(164), 421–423.
- Johnson-Laird, P. N.** (1983). *Mental Models*. Harvard University Press, Cambridge, MA.
- Jones, M. P.** and **Martin, J. H.** (1997). Contextual spelling correction using latent semantic analysis. In *ANLP 1997*, 166–173.
- Jones, R., McCallum, A., Nigam, K., and Riloff, E.** (1999). Bootstrapping for text learning tasks. In *IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*.
- Jones, T.** (2015). Toward a description of African American Vernacular English dialect regions using “Black Twitter”. *American Speech*, 90(4), 403–440.
- Joos, M.** (1950). Description of language design. *JASA*, 22, 701–708.
- Jordan, M.** (1986). Serial order: A parallel distributed processing approach. Tech. rep. ICS Report 8604, University of California, San Diego.
- Joshi, A. K.** (1985). Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?. In Dowty, D. R., Karttunen, L., and Zwicky, A. (Eds.), *Natural Language Parsing*, 206–250. Cambridge University Press.
- Joshi, A. K.** and **Hopely, P.** (1999). A parser from antiquity. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, 6–15. Cambridge University Press.
- Joshi, A. K.** and **Kuhn, S.** (1979). Centered logic: The role of entity centered sentence representation in natural language inferencing. In *IJCAI-79*, 435–439.
- Joshi, A. K.** and **Srinivas, B.** (1994). Disambiguation of super parts of speech (or supertags): Almost parsing. In *COLING-94*, 154–160.
- Joshi, A. K.** and **Weinstein, S.** (1981). Control of inference: Role of some aspects of discourse structure – centering. In *IJCAI-81*, 385–387.
- Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L.** (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *ACL 2017*.
- Joshi, M., Levy, O., Weld, D. S., and Zettlemoyer, L.** (2019). BERT for coreference resolution: Baselines and analysis. In *EMNLP 2019*.

- Joty**, S., Carenini, G., and Ng, R. T. (2015). CODRA: A novel discriminative framework for rhetorical analysis. *Computational Linguistics*, 41(3), 385–435.
- Jun**, S.-A. (Ed.). (2005). *Prosodic Typology and Transcription: A Unified Approach*. Oxford University Press.
- Jurafsky**, D. (2014). *The Language of Food*. W. W. Norton, New York.
- Jurafsky**, D., Chahuneau, V., Routledge, B. R., and Smith, N. A. (2014). Narrative framing of consumer sentiment in online restaurant reviews. *First Monday*, 19(4).
- Jurafsky**, D., Wooters, C., Tajchman, G., Segal, J., Stolcke, A., Fosler, E., and Morgan, N. (1994). The Berkeley restaurant project. In *ICSLP-94*, 2139–2142.
- Jurgens**, D. and Klapaftis, I. P. (2013). Semeval-2013 task 13: Word sense induction for graded and non-graded senses. In **SEM*, 290–299.
- Jurgens**, D., Tsvetkov, Y., and Jurafsky, D. (2017). Incorporating dialectal variability for socially equitable language identification. In *ACL 2017*, 51–57.
- Justeson**, J. S. and Katz, S. M. (1991). Co-occurrences of antonymous adjectives and their contexts. *Computational linguistics*, 17(1), 1–19.
- Kalchbrenner**, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Kalyanpur**, A., Boguraev, B. K., Patwardhan, S., Murdock, J. W., Lally, A., Welty, C., Prager, J. M., Coppola, B., Fokoue-Nkoutche, A., Zhang, L., Pan, Y., and Qiu, Z. M. (2012). Structured data and inference in deepqa. *IBM Journal of Research and Development*, 56(3/4), 10:1–10:14.
- Kameyama**, M. (1986). A property-sharing constraint in centering. In *ACL-86*, 200–206.
- Kamp**, H. (1981). A theory of truth and semantic representation. In Groenendijk, J., Janssen, T., and Stokhof, M. (Eds.), *Formal Methods in the Study of Language*, 189–222. Mathematical Centre, Amsterdam.
- Kane**, S. K., Morris, M. R., Paradiso, A., and Campbell, J. (2017). “at times avuncular and cantankerous, with the reflexes of a mongoose”: Understanding self-expression through augmentative and alternative communication devices. In *CSCW 2017*, 1166–1179.
- Kannan**, A. and Vinyals, O. (2016). Adversarial evaluation of dialogue models. In *NIPS 2016 Workshop on Adversarial Training*.
- Kaplan**, R. M. (1973). A general syntactic processor. In Rustin, R. (Ed.), *Natural Language Processing*, 193–241. Algorithmics Press.
- Karamanis**, N., Poesio, M., Mellish, C., and Oberlander, J. (2004). Evaluating centering-based metrics of coherence for text structuring using a reliably annotated corpus. In *ACL-04*.
- Karlsson**, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.). (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter.
- Karttunen**, L. (1969). Discourse referents. In *COLING-69*. Preprint No. 70.
- Karttunen**, L. (1999). Comments on Joshi. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, 16–18. Cambridge University Press.
- Kasami**, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA.
- Katz**, J. J. and Fodor, J. A. (1963). The structure of a semantic theory. *Language*, 39, 170–210.
- Kawamoto**, A. H. (1988). Distributed representations of ambiguous words and their resolution in connectionist networks. In Small, S. L., Cottrell, G. W., and Tanenhaus, M. (Eds.), *Lexical Ambiguity Resolution*, 195–228. Morgan Kaufman.
- Kay**, M. (1967). Experiments with a powerful parser. In *Proc. 2eme Conference Internationale sur le Traitement Automatique des Langues*.
- Kay**, M. (1973). The MIND system. In Rustin, R. (Ed.), *Natural Language Processing*, 155–188. Algorithmics Press.
- Kay**, M. (1982). Algorithm schemata and data structures in syntactic processing. In Allén, S. (Ed.), *Text Processing: Text Analysis and Generation, Text Typology and Attribution*, 327–358. Almqvist and Wiksell, Stockholm.
- Kay**, P. and Fillmore, C. J. (1999). Grammatical constructions and linguistic generalizations: The What’s X Doing Y? construction. *Language*, 75(1), 1–33.
- Kehler**, A. (1993). The effect of establishing coherence in ellipsis and anaphora resolution. In *ACL-93*, 62–69.
- Kehler**, A. (1994). Temporal relations: Reference or discourse coherence?. In *ACL-94*, 319–321.
- Kehler**, A. (1997a). Current theories of centering for pronoun interpretation: A critical evaluation. *Computational Linguistics*, 23(3), 467–475.
- Kehler**, A. (1997b). Probabilistic coreference in information extraction. In *EMNLP 1997*, 163–173.
- Kehler**, A. (2000). *Coherence, Reference, and the Theory of Grammar*. CSLI Publications.
- Kehler**, A., Appelt, D. E., Taylor, L., and Simma, A. (2004). The (non)utility of predicate-argument frequencies for pronoun interpretation. In *HLT-NAACL-04*.
- Kehler**, A. and Rohde, H. (2013). A probabilistic reconciliation of coherence-driven and centering-driven theories of pronoun interpretation. *Theoretical Linguistics*, 39(1-2), 1–37.
- Keller**, F. and Lapata, M. (2003). Using the web to obtain frequencies for unseen bigrams. *Computational Linguistics*, 29, 459–484.
- Kelly**, E. F. and Stone, P. J. (1975). *Computer Recognition of English Word Senses*. North-Holland.
- Kennedy**, C. and Boguraev, B. K. (1996). Anaphora for everyone: Pronominal anaphora resolution without a parser. In *COLING-96*, 113–118.
- Kiela**, D. and Clark, S. (2014). A systematic study of semantic vector space model parameters. In *Proceedings of the EACL 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*, 21–30.
- Kilgarriff**, A. and Rosenzweig, J. (2000). Framework and results for English SENSEVAL. *Computers and the Humanities*, 34, 15–48.
- Kim**, S. M. and Hovy, E. H. (2004). Determining the sentiment of opinions. In *COLING-04*.
- Kingma**, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR 2015*.
- Kintsch**, W. (1974). *The Representation of Meaning in Memory*. Wiley, New York.
- Kintsch**, W. and Van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological review*, 85(5), 363–394.
- Kipper**, K., Dang, H. T., and Palmer, M. (2000). Class-based construction of a verb lexicon. In *AAAI-00*, 691–696.
- Kiritchenko**, S. and Mohammad, S. M. (2017). Best-worst scaling more reliable than rating scales: A case study on sentiment intensity annotation. In *ACL 2017*, 465–470.
- Kiss**, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4), 485–525.

- Kleene, S. C.** (1951). Representation of events in nerve nets and finite automata. Tech. rep. RM-704, RAND Corporation. RAND Research Memorandum.
- Kleene, S. C.** (1956). Representation of events in nerve nets and finite automata. In Shannon, C. and McCarthy, J. (Eds.), *Automata Studies*, 3–41. Princeton University Press.
- Klein, D.** and Manning, C. D. (2001). Parsing and hypergraphs. In *IWPT-01*, 123–134.
- Klein, D.** and Manning, C. D. (2003a). A* parsing: Fast exact Viterbi parse selection. In *HLT-NAACL-03*.
- Klein, D.** and Manning, C. D. (2003b). Accurate unlexicalized parsing. In *HLT-NAACL-03*.
- Klein, S.** and Simmons, R. F. (1963). A computational approach to grammatical coding of English words. *Journal of the Association for Computing Machinery*, 10(3), 334–347.
- Kneser, R.** and Ney, H. (1995). Improved backoff for M-gram language modeling. In *ICASSP-95*, Vol. 1, 181–184.
- Knott, A.** and Dale, R. (1994). Using linguistic phenomena to motivate a set of coherence relations. *Discourse Processes*, 18(1), 35–62.
- Kocijan, V.**, Cretu, A.-M., Camburu, O.-M., Yordanov, Y., and Lukasiewicz, T. (2019). A surprisingly robust trick for the Winograd Schema Challenge. In *ACL 2019*.
- Kočiský, T.**, Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E. (2018). The NarrativeQA reading comprehension challenge. *TACL*, 6, 317–328.
- Koehn, P.**, Abney, S. P., Hirschberg, J., and Collins, M. (2000). Improving intonational phrasing with syntactic information. In *ICASSP-00*, 1289–1290.
- Koenig, W.**, Dunn, H. K., and Lacy, L. Y. (1946). The sound spectrograph. *JASA*, 18, 19–49.
- Krovetz, R.** (1993). Viewing morphology as an inference process. In *SIGIR-93*, 191–202.
- Kruskal, J. B.** (1983). An overview of sequence comparison. In Sankoff, D. and Kruskal, J. B. (Eds.), *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, 1–44. Addison-Wesley.
- Kudo, T.** and Matsumoto, Y. (2002). Japanese dependency analysis using cascaded chunking. In *CoNLL-02*, 63–69.
- Kudo, T.** and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *EMNLP 2018*, 66–71.
- Kullback, S.** and Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, 22, 79–86.
- Kumar, S.**, Jat, S., Saxena, K., and Talukdar, P. (2019). Zero-shot word sense disambiguation using sense definition embeddings. In *ACL 2019*, 5670–5681.
- Kummerfeld, J. K.** and Klein, D. (2013). Error-driven analysis of challenges in coreference resolution. In *EMNLP 2013*, 265–277.
- Kuno, S.** (1965). The predictive analyzer and a path elimination technique. *CACM*, 8(7), 453–462.
- Kuno, S.** and Oettinger, A. G. (1963). Multiple-path syntactic analyzer. In Popplewell, C. M. (Ed.), *Information Processing 1962: Proceedings of the IFIP Congress 1962*, 306–312. North-Holland. Reprinted in Grosz et al. (1986).
- Kupiec, J.** (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6, 225–242.
- Kurita, K.**, Vyas, N., Pareek, A., Black, A. W., and Tsvetkov, Y. (2019). Quantifying social biases in contextual word representations. In *1st ACL Workshop on Gender Bias for Natural Language Processing*.
- Kučera, H.** and Francis, W. N. (1967). *Computational Analysis of Present-Day American English*. Brown University Press, Providence, RI.
- Ladd, D. R.** (1996). *Intonational Phonology*. Cambridge Studies in Linguistics. Cambridge University Press.
- Ladefoged, P.** (1993). *A Course in Phonetics*. Harcourt Brace Jovanovich. (3rd ed.).
- Ladefoged, P.** (1996). *Elements of Acoustic Phonetics* (2nd Ed.). University of Chicago.
- Lafferty, J. D.**, McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*.
- Lafferty, J. D.**, Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.
- Lai, A.** and Tetreault, J. (2018). Discourse coherence in the wild: A dataset, evaluation and methods. In *SIGDIAL 18*, 214–223.
- Lakoff, G.** (1965). *On the Nature of Syntactic Irregularity*. Ph.D. thesis, Indiana University. Published as *Irregularity in Syntax*. Holt, Rinehart, and Winston, New York, 1970.
- Lakoff, G.** (1972a). Linguistics and natural logic. In Davidson, D. and Harman, G. (Eds.), *Semantics for Natural Language*, 545–665. D. Reidel.
- Lakoff, G.** (1972b). Structural complexity in fairy tales. In *The Study of Man*, 128–50. School of Social Sciences, University of California, Irvine, CA.
- Lakoff, G.** and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press, Chicago, IL.
- Lally, A.**, Prager, J. M., McCord, M. C., Boguraev, B. K., Patwardhan, S., Fan, J., Fodor, P., and Chu-Carroll, J. (2012). Question analysis: How Watson reads a clue. *IBM Journal of Research and Development*, 56(3/4), 2:1–2:14.
- Lample, G.**, Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016a). Neural architectures for named entity recognition. In *NAACL HLT 2016*.
- Lample, G.**, Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016b). Neural architectures for named entity recognition. In *NAACL HLT 2016*.
- Landauer, T. K.** (Ed.). (1995). *The Trouble with Computers: Usefulness, Usability, and Productivity*. MIT Press.
- Landauer, T. K.** and Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104, 211–240.
- Landes, S.**, Leacock, C., and Tengi, R. I. (1998). Building semantic concordances. In Fellbaum, C. (Ed.), *WordNet: An Electronic Lexical Database*, 199–216. MIT Press.
- Lang, J.** and Lapata, M. (2014). Similarity-driven semantic role induction via graph partitioning. *Computational Linguistics*, 40(3), 633–669.
- Lapata, M.** (2003). Probabilistic text structuring: Experiments with sentence ordering. In *ACL-03*, 545–552.
- Lapesa, G.** and Evert, S. (2014). A large scale evaluation of distributional semantic models: Parameters, interactions and model selection. *TACL*, 2, 531–545.
- Lappin, S.** and Leass, H. (1994). An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4), 535–561.
- Lari, K.** and Young, S. J. (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4, 35–56.

- Lascarides**, A. and Asher, N. (1993). Temporal interpretation, discourse relations, and common sense entailment. *Linguistics and Philosophy*, 16(5), 437–493.
- Lauscher**, A., Vulić, I., Ponti, E. M., Korhonen, A., and Glavaš, G. (2019). Informing unsupervised pretraining with external linguistic knowledge. arXiv preprint arXiv:1909.02339.
- LDC** (1995). COMLEX English Pronunciation Dictionary Version 0.2 (COMLEX 0.2). Linguistic Data Consortium.
- LeCun**, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4), 541–551.
- LeCun**, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *NIPS 1990*, 396–404.
- Lee**, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755), 788–791.
- Lee**, H., Chang, A., Peirsman, Y., Chambers, N., Surdeanu, M., and Jurafsky, D. (2013). Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4), 885–916.
- Lee**, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M., and Jurafsky, D. (2011). Stanford’s multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In *CoNLL-11*, 28–34.
- Lee**, H., Surdeanu, M., and Jurafsky, D. (2017a). A scaffolding approach to coreference resolution integrating statistical and rule-based models. *Natural Language Engineering*, 23(5), 733–762.
- Lee**, K., He, L., Lewis, M., and Zettlemoyer, L. (2017b). End-to-end neural coreference resolution. In *EMNLP 2017*.
- Lee**, K., He, L., and Zettlemoyer, L. (2018). Higher-order coreference resolution with coarse-to-fine inference. In *NAACL HLT 2018*.
- Lee**, K., Salant, S., Kwiatkowski, T., Parikh, A., Das, D., and Berant, J. (2017). Learning recurrent span representations for extractive question answering. In *arXiv 1611.01436*.
- Lehiste**, I. (Ed.) (1967). *Readings in Acoustic Phonetics*. MIT Press.
- Lehnert**, W. G., Cardie, C., Fisher, D., Riloff, E., and Williams, R. (1991). Description of the CIRCUS system as used for MUC-3. In Sundheim, B. (Ed.), *MUC-3*, 223–233.
- Lemon**, O., Georgila, K., Henderson, J., and Stuttle, M. (2006). An ISU dialogue system exhibiting reinforcement learning of dialogue policies: Generic slot-filling in the TALK car system. In *EACL-06*.
- Lengerich**, B., Maas, A., and Potts, C. (2018). Retrofitting distributional embeddings to knowledge graphs with functional relations. In *Proceedings of the 27th International Conference on Computational Linguistics*, 2423–2436.
- Lesk**, M. E. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th International Conference on Systems Documentation*, 24–26.
- Leuski**, A. and Traum, D. (2011). NPCEditor: Creating virtual human dialogue using information retrieval techniques. *AI Magazine*, 32(2), 42–56.
- Levenshtein**, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8), 707–710. Original in *Doklady Akademii Nauk SSSR* 163(4): 845–848 (1965).
- Levesque**, H. (2011). The Winograd Schema Challenge. In *Logical Formalizations of Commonsense Reasoning —Papers from the AAAI 2011 Spring Symposium (SS-11-06)*.
- Levesque**, H., Davis, E., and Morgenstern, L. (2012). The Winograd Schema Challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.
- Levesque**, H. J., Cohen, P. R., and Nunes, J. H. T. (1990). On acting together. In *AAAI-90*, 94–99. Morgan Kaufmann.
- Levin**, B. (1977). Mapping sentences to case frames. Tech. rep. 167, MIT AI Laboratory. AI Working Paper 143.
- Levin**, B. (1993). *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press.
- Levin**, B. and Rappaport Hovav, M. (2005). *Argument Realization*. Cambridge University Press.
- Levin**, E., Pieraccini, R., and Eckert, W. (2000). A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on Speech and Audio Processing*, 8, 11–23.
- Levine**, Y., Lenz, B., Dagan, O., Padnos, D., Sharir, O., Shalev-Shwartz, S., Shashua, A., and Shoham, Y. (2019). Sensebert: Driving some sense into bert. arXiv preprint arXiv:1908.05646.
- Levinson**, S. C. (1983). *Conversational Analysis*, chap. 6. Cambridge University Press.
- Levow**, G.-A. (1998). Characterizing and recognizing spoken corrections in human-computer dialogue. In *COLING-ACL*, 736–742.
- Levy**, O. and Goldberg, Y. (2014a). Dependency-based word embeddings. In *ACL 2014*.
- Levy**, O. and Goldberg, Y. (2014b). Linguistic regularities in sparse and explicit word representations. In *CoNLL-14*.
- Levy**, O. and Goldberg, Y. (2014c). Neural word embedding as implicit matrix factorization. In *NIPS 14*, 2177–2185.
- Levy**, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3, 211–225.
- Lewis**, M. and Steedman, M. (2014). A* ccg parsing with a supertag-factored model.. In *EMNLP*, 990–1000.
- Li**, A., Zheng, F., Byrne, W., Fung, P., Kamm, T., Yi, L., Song, Z., Ruhi, U., Venkataramani, V., and Chen, X. (2000). CASS: A phonetically transcribed corpus of Mandarin spontaneous speech. In *IICSLP-00*, 485–488.
- Li**, J., Chen, X., Hovy, E. H., and Jurafsky, D. (2015). Visualizing and understanding neural models in NLP. In *NAACL HLT 2015*.
- Li**, J., Galley, M., Brockett, C., Gao, J., and Dolan, B. (2016). A diversity-promoting objective function for neural conversation models. In *NAACL HLT 2016*.
- Li**, J. and Jurafsky, D. (2017). Neural net models of open-domain discourse coherence. In *EMNLP 2017*, 198–209.
- Li**, J., Li, R., and Hovy, E. H. (2014). Recursive deep models for discourse parsing. In *EMNLP 2014*, 2061–2069.
- Li**, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016). Deep reinforcement learning for dialogue generation. In *EMNLP 2016*.
- Li**, J., Monroe, W., Shi, T., Ritter, A., and Jurafsky, D. (2017). Adversarial learning for neural dialogue generation. In *EMNLP 2017*.
- Li**, Q., Li, T., and Chang, B. (2016). Discourse parsing with attention-based hierarchical neural networks. In *EMNLP 2016*, 362–371.
- Li**, X. and Roth, D. (2002). Learning question classifiers. In *COLING-02*, 556–562.

- Li, X.** and Roth, D. (2005). Learning question classifiers: The role of semantic information. *Journal of Natural Language Engineering*, 11(4).
- Li, X., Meng, Y., Sun, X., Han, Q., Yuan, A., and Li, J.** (2019). Is word segmentation necessary for deep learning of Chinese representations?. In *ACL 2019*, 3242–3252.
- Lieberman, A. M., Delattre, P. C., and Cooper, F. S.** (1952). The role of selected stimulus variables in the perception of the unvoiced stop consonants. *American Journal of Psychology*, 65, 497–516.
- Lin, D.** (1995). A dependency-based method for evaluating broad-coverage parsers. In *IJCAI-95*, 1420–1425.
- Lin, D.** (2003). Dependency-based evaluation of minipar. In *Workshop on the Evaluation of Parsing Systems*.
- Lin, J.** (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems*, 25(2).
- Lin, Y., Michel, J.-B., Lieberman Aiden, E., Orwant, J., Brockman, W., and Petrov, S.** (2012). Syntactic annotations for the google books ngram corpus. In *ACL 2012*, 169–174.
- Lin, Z., Madotto, A., Shin, J., Xu, P., and Fung, P.** (2019). MoEL: Mixture of empathetic listeners. <http://arxiv.org/abs/1908.07687>.
- Lin, Z., Kan, M.-Y., and Ng, H. T.** (2009). Recognizing implicit discourse relations in the penn discourse treebank. In *EMNLP-09*, 343–351.
- Lin, Z., Ng, H. T., and Kan, M.-Y.** (2011). Automatically evaluating text coherence using discourse relations. In *ACL 2011*, 997–1006.
- Lin, Z., Ng, H. T., and Kan, M.-Y.** (2014). A pdtb-styled end-to-end discourse parser. *Natural Language Engineering*, 20(2), 151–184.
- Lindsey, R.** (1963). Inferential memory as the basis of machines which understand natural language. In Feigenbaum, E. and Feldman, J. (Eds.), *Computers and Thought*, 217–233. McGraw Hill.
- Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fernandez, R., Amir, S., Marujo, L., and Luis, T.** (2015a). Finding function in form: Compositional character models for open vocabulary word representation. In *EMNLP 2015*, 1520–1530.
- Ling, X., Singh, S., and Weld, D. S.** (2015b). Design challenges for entity linking. *TACL*, 3, 315–328.
- Lison, P. and Tiedemann, J.** (2016). Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. In *LREC-16*.
- Litman, D. J.** (1985). *Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues*. Ph.D. thesis, University of Rochester, Rochester, NY.
- Litman, D. J. and Allen, J.** (1987). A plan recognition model for subdialogues in conversation. *Cognitive Science*, 11, 163–200.
- Litman, D. J., Swerts, M., and Hirschberg, J.** (2000). Predicting automatic speech recognition performance using prosodic cues. In *NAACL 2000*.
- Litman, D. J., Walker, M. A., and Kearns, M.** (1999). Automatic detection of poor speech recognition at the dialogue level. In *ACL-99*, 309–316.
- Liu, B.** and Zhang, L. (2012). A survey of opinion mining and sentiment analysis. In Aggarwal, C. C. and Zhai, C. (Eds.), *Mining text data*, 415–464. Springer.
- Liu, C.-W., Lowe, R. T., Serban, I. V., Noseworthy, M., Charlin, L., and Pineau, J.** (2016). How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. In *EMNLP 2016*.
- Lochbaum, K. E., Grosz, B. J., and Sidner, C. L.** (2000). Discourse structure and intention recognition. In Dale, R., Moisl, H., and Somers, H. L. (Eds.), *Handbook of Natural Language Processing*. Marcel Dekker.
- Logeswaran, L., Lee, H., and Radev, D.** (2018). Sentence ordering and coherence modeling using recurrent neural networks. In *AAAI-18*.
- Louis, A.** and Nenkova, A. (2012). A coherence model based on syntactic patterns. In *EMNLP 2012*, 1157–1168.
- Loureiro, D.** and Jorge, A. (2019). Language modelling makes sense: Propagating representations through WordNet for full-coverage word sense disambiguation. In *ACL 2019*, 5682–5691.
- Louviere, J. J., Flynn, T. N., and Marley, A. A. J.** (2015). *Best-worst scaling: Theory, methods and applications*. Cambridge University Press.
- Lovins, J. B.** (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2), 9–13.
- Lowe, R. T., Noseworthy, M., Serban, I. V., Angelard-Gontier, N., Bengio, Y., and Pineau, J.** (2017a). Towards an automatic Turing test: Learning to evaluate dialogue responses. In *ACL 2017*.
- Lowe, R. T., Pow, N., Serban, I. V., Charlin, L., Liu, C.-W., and Pineau, J.** (2017b). Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue & Discourse*, 8(1), 31–65.
- Luhn, H. P.** (1957). A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4), 309–317.
- Lui, M.** and Baldwin, T. (2011). Cross-domain feature selection for language identification. In *IJCNLP-11*, 553–561.
- Lui, M.** and Baldwin, T. (2012). langid.py: An off-the-shelf language identification tool. In *ACL 2012*, 25–30.
- Luo, F., Liu, T., He, Z., Xia, Q., Sui, Z., and Chang, B.** (2018a). Leveraging gloss knowledge in neural word sense disambiguation by hierarchical co-attention. In *EMNLP 2018*, 1402–1411.
- Luo, F., Liu, T., Xia, Q., Chang, B., and Sui, Z.** (2018b). Incorporating glosses into neural word sense disambiguation. In *ACL 2018*, 2473–2482.
- Luo, X.** (2005). On coreference resolution performance metrics. In *Proceedings of HLT-EMNLP 2005*, 25–32.
- Luo, X.** and Pradhan, S. (2016). Evaluation metrics. In Poesio, M., Stuckardt, R., and Versley, Y. (Eds.), *Anaphora resolution: Algorithms, resources, and applications*, 141–163. Springer.
- Luo, X., Pradhan, S., Recasens, M., and Hovy, E. H.** (2014). An extension of blanc to system mentions. In *ACL 2014*, Vol. 2014, p. 24.
- Luong, T., Pham, H., and Manning, C. D.** (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1412–1421.
- Lyons, J.** (1977). *Semantics*. Cambridge University Press.
- Lyons, R. G.** (2004). *Understanding Digital Signal Processing*. Prentice Hall. (2nd. ed.).
- Ma, X.** and Hovy, E. H. (2016). End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *ACL 2016*.
- Madhu, S.** and Lytel, D. (1965). A figure of merit technique for the resolution of non-grammatical ambiguity. *Mechanical Translation*, 8(2), 9–13.
- Magerman, D. M.** (1994). *Natural Language Parsing as Statistical Pattern Recognition*. Ph.D. thesis, University of Pennsylvania.

- Magerman, D. M.** (1995). Statistical decision-tree models for parsing. In *ACL-95*, 276–283.
- Magerman, D. M.** and **Marcus, M. P.** (1991). Pearl: A probabilistic chart parser. In *EACL-91*.
- Mairesse, F.** and **Walker, M. A.** (2008). Trainable generation of big-five personality styles through data-driven parameter estimation. In *ACL-08*.
- Manandhar, S.**, Klapftis, I. P., Dligach, D., and Pradhan, S. (2010). SemEval-2010 task 14: Word sense induction & disambiguation. In *SemEval-2010*, 63–68.
- Mann, W. C.** and **Thompson, S. A.** (1987). Rhetorical structure theory: A theory of text organization. Tech. rep. RS-87-190, Information Sciences Institute.
- Manning, C. D.** (2011). Part-of-speech tagging from 97% to 100%: Is it time for some linguistics?. In *CICLing 2011*, 171–189.
- Manning, C. D.**, Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge.
- Manning, C. D.** and **Schütze, H.** (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.
- Manning, C. D.**, Surdeanu, M., Bauer, J., Finkel, J., Bethard, S., and McClosky, D. (2014). The stanford CoreNLP natural language processing toolkit. In *ACL 2014*, 55–60.
- Marcu, D.** (1997). The rhetorical parsing of natural language texts. In *ACL/EACL-97*.
- Marcu, D.** (1999). A decision-based approach to rhetorical parsing. In *ACL-99*, 365–372.
- Marcu, D.** (2000a). The rhetorical parsing of unrestricted texts: A surface-based approach. *Computational Linguistics*, 26(3), 395–448.
- Marcu, D.** (Ed.). (2000b). *The Theory and Practice of Discourse Parsing and Summarization*. MIT Press.
- Marcu, D.** and Echihabi, A. (2002). An unsupervised approach to recognizing discourse relations. In *ACL-02*, 368–375.
- Marcus, M. P.** (1980). *A Theory of Syntactic Recognition for Natural Language*. MIT Press.
- Marcus, M. P.** (1990). Summary of session 9: Automatic acquisition of linguistic structure. In *Proceedings DARPA Speech and Natural Language Workshop*, 249–250.
- Marcus, M. P.**, Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating predicate argument structure. In *ARPA Human Language Technology Workshop*, 114–119. Morgan Kaufmann.
- Marcus, M. P.**, Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2), 313–330.
- Markov, A. A.** (1913). Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7, 153–162.
- Markov, A. A.** (2006). Classical text in translation: A. A. Markov, an example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. *Science in Context*, 19(4), 591–600. Translated by David Link.
- Maron, M. E.** (1961). Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM)*, 8(3), 404–417.
- Márquez, L.**, Carreras, X., Litkowski, K. C., and Stevenson, S. (2008). Semantic role labeling: An introduction to the special issue. *Computational linguistics*, 34(2), 145–159.
- Marshall, I.** (1983). Choice of grammatical word-class without Global syntactic analysis: Tagging words in the LOB corpus. *Computers and the Humanities*, 17, 139–150.
- Marshall, I.** (1987). Tag selection using probabilistic methods. In Gar-side, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, 42–56. Longman.
- Martin, J. H.** (1986). The acquisition of polysemy. In *ICML 1986*, 198–204.
- Martschat, S.** and Strube, M. (2014). Recall error analysis for coreference resolution. In *EMNLP 2014*, 2070–2081.
- Martschat, S.** and Strube, M. (2015). Latent structures for coreference resolution. *TACL*, 3, 405–418.
- Masterman, M.** (1957). The thesaurus in syntax and semantics. *Mechanical Translation*, 4(1), 1–2.
- Mathis, D. A.** and Mozer, M. C. (1995). On the computational utility of consciousness. In Tesauro, G., Touretzky, D. S., and Alspector, J. (Eds.), *Advances in Neural Information Processing Systems VII*. MIT Press.
- McCallum, A.**, Freitag, D., and Pereira, F. C. N. (2000). Maximum entropy Markov models for information extraction and segmentation. In *ICML 2000*, 591–598.
- McCallum, A.** and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *AAAI/ICML-98 Workshop on Learning for Text Categorization*, 41–48.
- McCarthy, J. F.** and **Lehnert, W. G.** (1995). Using decision trees for coreference resolution. In *IJCAI-95*, 1050–1055.
- McCawley, J. D.** (1968). The role of semantics in a grammar. In Bach, E. W. and Harms, R. T. (Eds.), *Universals in Linguistic Theory*, 124–169. Holt, Rinehart & Winston.
- McCawley, J. D.** (1993). *Everything that Linguists Have Always Wanted to Know about Logic* (2nd Ed.). University of Chicago Press, Chicago, IL.
- McCawley, J. D.** (1998). *The Syntactic Phenomena of English*. University of Chicago Press.
- McClelland, J. L.** and Elman, J. L. (1986). The TRACE model of speech perception. *Cognitive Psychology*, 18, 1–86.
- McClelland, J. L.**, Rumelhart, D. E., and The PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 2: *Psychological and Biological Models*. Bradford Books (MIT Press).
- McCulloch, W. S.** and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115–133. Reprinted in *Neurocomputing: Foundations of Research*, ed. by J. A. Anderson and E. Rosenfeld. MIT Press 1988.
- McDonald, R.**, Crammer, K., and Pereira, F. C. N. (2005). Online large-margin training of dependency parsers. In *ACL-05*, 91–98.
- McDonald, R.** and Nivre, J. (2011). Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1), 197–230.
- McDonald, R.**, Pereira, F. C. N., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *HLT-EMNLP-05*.
- McGuiness, D. L.** and van Harmelen, F. (2004). OWL web ontology overview. Tech. rep. 20040210, World Wide Web Consortium.
- Mehl, M. R.**, Gosling, S. D., and Pennebaker, J. W. (2006). Personality in its natural habitat: manifestations and implicit folk theories of personality in daily life.. *Journal of Personality and Social Psychology*, 90(5).
- Melamud, O.**, Goldberger, J., and Da-gan, I. (2016). context2vec: Learning generic context embedding with bidirectional LSTM. In *CoNLL-16*, 51–61.

- Mel'čuk, I. A.** (1988). *Dependency Syntax: Theory and Practice*. State University of New York Press.
- Merialdo, B.** (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2), 155–172.
- Mesgar, M.** and Strube, M. (2016). Lexical coherence graph modeling using word embeddings. In *ACL 2016*, 1414–1423.
- Metsis, V.**, Androutsopoulos, I., and Palioras, G. (2006). Spam filtering with naive bayes—which naive bayes?. In *CEAS*, 27–28.
- Meyers, A.**, Reeves, R., Macleod, C., Szekely, R., Zielinska, V., Young, B., and Grishman, R. (2004). The nombank project: An interim report. In *Proceedings of the NAACL/HLT Workshop: Frontiers in Corpus Annotation*.
- Mihalcea, R.** (2007). Using wikipedia for automatic word sense disambiguation. In *NAACL-HLT 07*, 196–203.
- Mihalcea, R.** and Csomai, A. (2007). Wikify!: linking documents to encyclopedic knowledge. In *CIKM 2007*, 233–242.
- Mihalcea, R.** and Moldovan, D. (2001). Automatic generation of a coarse grained WordNet. In *NAACL Workshop on WordNet and Other Lexical Resources*.
- Mikheev, A.**, Moens, M., and Grover, C. (1999). Named entity recognition without gazetteers. In *EACL-99*, 1–8.
- Mikolov, T.** (2012). *Statistical language models based on neural networks*. Ph.D. thesis, Ph. D. thesis, Brno University of Technology.
- Mikolov, T.**, Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *ICLR 2013*.
- Mikolov, T.**, Karafiat, M., Burget, L., Černocky, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH 2010*, 1045–1048.
- Mikolov, T.**, Kombrink, S., Burget, L., Černocky, J. H., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *ICASSP-11*, 5528–5531.
- Mikolov, T.**, Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *NIPS 13*, 3111–3119.
- Mikolov, T.**, Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In *NAACL HLT 2013*, 746–751.
- Miller, C. A.** (1998). Pronunciation modeling in speech synthesis. Tech. rep. IRCS 98–09, University of Pennsylvania Institute for Research in Cognitive Science, Philadelphia, PA.
- Miller, G. A.** and Nicely, P. E. (1955). An analysis of perceptual confusions among some English consonants. *JASA*, 27, 338–352.
- Miller, G. A.** and Charles, W. G. (1991). Contextual correlates of semantics similarity. *Language and Cognitive Processes*, 6(1), 1–28.
- Miller, G. A.** and Chomsky, N. (1963). Finitary models of language users. In Luce, R. D., Bush, R. R., and Galanter, E. (Eds.), *Handbook of Mathematical Psychology*, Vol. II, 419–491. John Wiley.
- Miller, G. A.**, Leacock, C., Tengi, R. I., and Bunker, R. T. (1993). A semantic concordance. In *Proceedings ARPA Workshop on Human Language Technology*, 303–308.
- Miller, G. A.** and Selfridge, J. A. (1950). Verbal context and the recall of meaningful material. *American Journal of Psychology*, 63, 176–185.
- Miller, S.**, Bobrow, R. J., Ingria, R., and Schwartz, R. (1994). Hidden understanding models of natural language. In *ACL-94*, 25–32.
- Milne, D.** and Witten, I. H. (2008). Learning to link with wikipedia. In *CIKM 2008*, 509–518.
- Miltsakaki, E.**, Prasad, R., Joshi, A. K., and Webber, B. L. (2004). The Penn Discourse Treebank. In *LREC-04*.
- Minsky, M.** (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1), 8–30.
- Minsky, M.** (1974). A framework for representing knowledge. Tech. rep. 306, MIT AI Laboratory. Memo 306.
- Minsky, M.** and Papert, S. (1969). *Perceptrons*. MIT Press.
- Mintz, M.**, Bills, S., Snow, R., and Jurafsky, D. (2009). Distant supervision for relation extraction without labeled data. In *ACL IJCNLP 2009*.
- Mitkov, R.** (2002). *Anaphora Resolution*. Longman.
- Miwa, M.** and Bansal, M. (2016). End-to-end relation extraction using lstms on sequences and tree structures. In *ACL 2016*, 1105–1116.
- Mohammad, S. M.** (2018a). Obtaining reliable human ratings of valence, arousal, and dominance for 20,000 english words. In *ACL 2018*.
- Mohammad, S. M.** (2018b). Word affect intensities. In *LREC-18*.
- Mohammad, S. M.** and Turney, P. D. (2013). Crowdsourcing a word-emotion association lexicon. *Computational Intelligence*, 29(3), 436–465.
- Monroe, B. L.**, Colaresi, M. P., and Quinn, K. M. (2008). Fightin'words: Lexical feature selection and evaluation for identifying the content of political conflict. *Political Analysis*, 16(4), 372–403.
- Montague, R.** (1973). The proper treatment of quantification in ordinary English. In Thomason, R. (Ed.), *Formal Philosophy: Selected Papers of Richard Montague*, 247–270. Yale University Press, New Haven, CT.
- Monz, C.** (2004). Minimal span weighting retrieval for question answering. In *SIGIR Workshop on Information Retrieval for Question Answering*, 23–30.
- Moosavi, N. S.** and Strube, M. (2016). Which coreference evaluation metric do you trust? a proposal for a link-based entity aware metric. In *ACL 2016*, 632–642.
- Morey, M.**, Muller, P., and Asher, N. (2017). How much progress have we made on RST discourse parsing? a replication study of recent results on the rst-dt. In *EMNLP 2017*.
- Morgan, A. A.**, Hirschman, L., Colosimo, M., Yeh, A. S., and Colombe, J. B. (2004). Gene name identification and normalization using a model organism database. *Journal of Biomedical Informatics*, 37(6), 396–410.
- Morgan, N.** and Bourlard, H. (1989). Generalization and parameter estimation in feedforward nets: Some experiments. In *Advances in neural information processing systems*, 630–637.
- Morgan, N.** and Bourlard, H. (1990). Continuous speech recognition using multilayer perceptrons with hidden markov models. In *ICASSP-90*, 413–416.
- Morris, J.** and Hirst, G. (1991). Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, 17(1), 21–48.
- Morris, W.** (Ed.). (1985). *American Heritage Dictionary* (2nd College Edition Ed.). Houghton Mifflin.
- Mosteller, F.** and Wallace, D. L. (1963). Inference in an authorship problem: A comparative study of discrimination methods applied to the authorship of the disputed federalist papers. *Journal of the American Statistical Association*, 58(302), 275–309.
- Mosteller, F.** and Wallace, D. L. (1964). *Inference and Disputed Authorship: The Federalist*. Springer-Verlag. A second edition appeared in 1984 as *Applied Bayesian and Classical Inference*.
- Mrkšić, N.**, Ó Séaghdha, D., Wen, T.-H., Thomson, B., and Young, S.

- (2017). Neural belief tracker: Data-driven dialogue state tracking. In *ACL 2017*, 1777–1788.
- Mrkšić**, N., Séaghdha, D. Ó., Thompson, B., Gašić, M., Rojas-Barahona, L. M., Su, P.-H., Vandyke, D., Wen, T.-H., and Young, S. (2016). Counterfitting word vectors to linguistic constraints. In *NAACL HLT 2016*, 142–148.
- Muller**, P., Braud, C., and Morey, M. (2019). ToNy: Contextual embeddings for accurate multilingual discourse segmentation of full documents. In *Proceedings of the Workshop on Discourse Relation Parsing and Treebanking 2019*, 115–124.
- Murdock**, J. W., Fan, J., Lally, A., Shima, H., and Boguraev, B. K. (2012a). Textual evidence gathering and analysis. *IBM Journal of Research and Development*, 56(3/4), 8:1–8:14.
- Murdock**, J. W., Kalyanpur, A., Welty, C., Fan, J., Ferrucci, D. A., Gondek, D. C., Zhang, L., and Kanayama, H. (2012b). Typing candidate answers using type coercion. *IBM Journal of Research and Development*, 56(3/4), 7:1–7:13.
- Murphy**, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT press.
- Musi**, E., Stede, M., Kriese, L., Mureşan, S., and Rocci, A. (2018). A multi-layer annotated corpus of argumentative text: From argument schemes to discourse relations. In *LREC-18*.
- Myers**, G. (1992). “in this paper we report...”: Speech acts and scientific facts. *Journal of Pragmatics*, 17(4), 295–313.
- Nádas**, A. (1984). Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Transactions on Acoustics, Speech, Signal Processing*, 32(4), 859–861.
- Nagata**, M. and Morimoto, T. (1994). First steps toward statistical modeling of dialogue to predict the speech act type of the next utterance. *Speech Communication*, 15, 193–203.
- Nash-Webber**, B. L. (1975). The role of semantics in automatic speech understanding. In Bobrow, D. G. and Collins, A. (Eds.), *Representation and Understanding*, 351–382. Academic Press.
- Naur**, P., Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijnagaarden, A., and Woodger, M. (1960). Report on the algorithmic language ALGOL 60. *CACM*, 3(5), 299–314. Revised in CACM 6:1, 1–17, 1963.
- Navigli**, R. (2006). Meaningful clustering of senses helps boost word sense disambiguation performance. In *COLING/ACL 2006*, 105–112.
- Navigli**, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2).
- Navigli**, R. (2016). Chapter 20. ontologies. In Mitkov, R. (Ed.), *The Oxford handbook of computational linguistics*. Oxford University Press.
- Navigli**, R. and Ponzetto, S. P. (2012). BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193, 217–250.
- Navigli**, R. and Vannella, D. (2013). Semeval-2013 task 11: Word sense induction & disambiguation within an end-user application. In **SEM*, 193–201.
- Navayak**, N., Hakkani-Tür, D., Walker, M. A., and Heck, L. P. (2017). To plan or not to plan? discourse planning in slot-value informed sequence to sequence models for language generation.. In *INTERSPEECH-17*, 3339–3343.
- Needleman**, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino-acid sequence of two proteins. *Journal of Molecular Biology*, 48, 443–453.
- Neff**, G. and Nagy, P. (2016). Talking to bots: Symbiotic agency and the case of Tay. *International Journal of Communication*, 10, 4915–4931.
- Ney**, H. (1991). Dynamic programming parsing for context-free grammars in continuous speech recognition. *IEEE Transactions on Signal Processing*, 39(2), 336–340.
- Ng**, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS 14*, 841–848.
- Ng**, H. T., Teo, L. H., and Kwan, J. L. P. (2000). A machine learning approach to answering questions for reading comprehension tests. In *EMNLP 2000*, 124–132.
- Ng**, V. (2004). Learning noun phrase anaphoricity to improve coreference resolution: Issues in representation and optimization. In *ACL-04*.
- Ng**, V. (2005a). Machine learning for coreference resolution: From local classification to global ranking. In *ACL-05*.
- Ng**, V. (2005b). Supervised ranking for pronoun resolution: Some recent improvements. In *AAAI-05*.
- Ng**, V. (2010). Supervised noun phrase coreference research: The first fifteen years. In *ACL 2010*, 1396–1411.
- Ng**, V. (2017). Machine learning for entity coreference resolution: A retrospective look at two decades of research. In *AAAI-17*.
- Ng**, V. and Cardie, C. (2002a). Identifying anaphoric and non-anaphoric noun phrases to improve coreference resolution. In *COLING-02*.
- Ng**, V. and Cardie, C. (2002b). Improving machine learning approaches to coreference resolution. In *ACL-02*.
- Nguyen**, D. T. and Joty, S. (2017). A neural local coherence model. In *ACL 2017*, 1320–1330.
- Nguyen**, K. A., Schulte im Walde, S., and Vu, N. T. (2016). Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. In *ACL 2016*, 454–459.
- Nie**, A., Bennett, E., and Goodman, N. (2019). Dissent: Learning sentence representations from explicit discourse relations. In *ACL 2019*, 4497–4510.
- Nielsen**, J. (1992). The usability engineering life cycle. *IEEE Computer*, 25(3), 12–22.
- Nielsen**, M. A. (2015). *Neural networks and Deep learning*. Determination Press USA.
- Nigam**, K., Lafferty, J. D., and McCallum, A. (1999). Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, 61–67.
- Nilsson**, J., Riedel, S., and Yuret, D. (2007). The conll 2007 shared task on dependency parsing. In *Proceedings of the CoNLL shared task session of EMNLP-CoNLL*, 915–932. sn.
- Nissim**, M., Dingare, S., Carletta, J., and Steedman, M. (2004). An annotation scheme for information status in dialogue. In *LREC-04*.
- NIST** (1990). TIMIT Acoustic-Phonetic Continuous Speech Corpus. National Institute of Standards and Technology Speech Disc 1-1.1. NIST Order No. PB91-505065.
- NIST** (2005). Speech recognition scoring toolkit (sctk) version 2.1. <http://www.nist.gov/speech/tools/>.
- Nivre**, J. (2007). Incremental non-projective dependency parsing. In *NAACL-HLT 07*.
- Nivre**, J. (2003). An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*.
- Nivre**, J. (2006). *Inductive Dependency Parsing*. Springer.

- Nivre, J.** (2009). Non-projective dependency parsing in expected linear time. In *ACL IJCNLP 2009*, 351–359.
- Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic̄, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D.** (2016a). Universal Dependencies v1: A multilingual treebank collection. In *LREC*.
- Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic̄, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D.** (2016b). Universal Dependencies v1: A multilingual treebank collection. In *LREC-16*.
- Nivre, J., Hall, J., Nilsson, J., Chaney, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E.** (2007). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(02), 95–135.
- Nivre, J. and Nilsson, J.** (2005). Pseudo-projective dependency parsing. In *ACL-05*, 99–106.
- Nivre, J. and Scholz, M.** (2004). Deterministic dependency parsing of english text. In *COLING-04*, p. 64.
- Niwa, Y. and Nitta, Y.** (1994). Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *ACL-94*, 304–309.
- Noreen, E. W.** (1989). *Computer Intensive Methods for Testing Hypothesis*. Wiley.
- Norman, D. A.** (1988). *The Design of Everyday Things*. Basic Books.
- Norman, D. A. and Rumelhart, D. E.** (1975). *Explorations in Cognition*. Freeman.
- Norvig, P.** (1991). Techniques for automatic memoization with applications to context-free parsing. *Computational Linguistics*, 17(1), 91–98.
- Nosek, B. A., Banaji, M. R., and Greenwald, A. G.** (2002a). Harvesting implicit group attitudes and beliefs from a demonstration web site. *Group Dynamics: Theory, Research, and Practice*, 6(1), 101.
- Nosek, B. A., Banaji, M. R., and Greenwald, A. G.** (2002b). Math=male, me=female, therefore math≠ me. *Journal of personality and social psychology*, 83(1), 44.
- O'Connor, B., Krieger, M., and Ahn, D.** (2010). Tweetmotif: Exploratory search and topic summarization for twitter. In *ICWSM*.
- Oravecz, C. and Dienes, P.** (2002). Efficient stochastic part-of-speech tagging for Hungarian. In *LREC-02*, 710–717.
- Osgood, C. E., Suci, G. J., and Tannenbaum, P. H.** (1957). *The Measurement of Meaning*. University of Illinois Press.
- O'Shaughnessy, D.** (2000). *Speech Communications: Human and Machine*. IEEE Press, New York. 2nd ed.
- Ostendorf, M. and Veilleux, N.** (1994). A hierarchical stochastic model for automatic prediction of prosodic boundary location. *Computational Linguistics*, 20(1), 27–54.
- Packard, D. W.** (1973). Computer-assisted morphological analysis of ancient Greek. In Zampolli, A. and Calzolari, N. (Eds.), *Computational and Mathematical Linguistics: Proceedings of the International Conference on Computational Linguistics*, 343–355. Leo S. Olschki.
- Palmer, D.** (2012). Text preprocessing. In Indurkha, N. and Damerau, F. J. (Eds.), *Handbook of Natural Language Processing*, 9–30. CRC Press.
- Palmer, M., Babko-Malaya, O., and Dang, H. T.** (2004). Different sense granularities for different applications. In *HLT-NAACL Workshop on Scalable Natural Language Understanding*, 49–56.
- Palmer, M., Dang, H. T., and Fellbaum, C.** (2006). Making fine-grained and coarse-grained sense distinctions, both manually and automatically. *Natural Language Engineering*, 13(2), 137–163.
- Palmer, M., Gildea, D., and Xue, N.** (2010). Semantic role labeling. *Synthesis Lectures on Human Language Technologies*, 3(1), 1–103.
- Palmer, M., Kingsbury, P., and Gildea, D.** (2005). The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1), 71–106.
- Pang, B. and Lee, L.** (2008). Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2), 1–135.
- Pang, B., Lee, L., and Vaithyanathan, S.** (2002). Thumbs up? Sentiment classification using machine learning techniques. In *EMNLP 2002*, 79–86.
- Paolino, J.** (2017). Google Home vs Alexa: Two simple user experience design gestures that delighted a female user. In *Medium*. Jan 4, 2017. <https://medium.com/startup-grid/google-home-vs-alexa-56e26f69ac77>
- Park, J. and Cardie, C.** (2014). Identifying appropriate support for propositions in online user comments. In *Proceedings of the first workshop on argumentation mining*, 29–38.
- Parsons, T.** (1990). *Events in the Semantics of English*. MIT Press.
- Partee, B. H. (Ed.)** (1976). *Montague Grammar*. Academic Press.
- Pasca, M.** (2003). *Open-Domain Question Answering from Large Text Collections*. CSLI.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A.** (2017). Automatic differentiation in pytorch. In *NIPS-W*.
- Pearl, C.** (2017). *Designing Voice User Interfaces: Principles of Conversational Experiences*. O'Reilly.
- Pedersen, T. and Bruce, R.** (1997). Distinguishing word senses in untagged text. In *EMNLP 1997*.
- Peldszus, A. and Stede, M.** (2013). From argument diagrams to argumentation mining in texts: A survey. *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, 7(1), 1–31.
- Peldszus, A. and Stede, M.** (2016). An annotated corpus of argumentative microtexts. In *Proceedings of the 1st European Conference on Argumentation*, 801–816.
- Peng, N., Poon, H., Quirk, C., Toutanova, K., and Yih, W.-t.** (2017). Cross-sentence n-ary relation extraction with graph LSTMs. *TACL*, 5, 101–115.
- Penn, G. and Kiparsky, P.** (2012). On Pāṇini and the generative capacity of contextualized replacement systems. In *COLING-12*, 943–950.
- Pennebaker, J. W., Booth, R. J., and Francis, M. E.** (2007). *Linguistic Inquiry and Word Count: LIWC 2007*. Austin, TX.
- Pennebaker, J. W. and King, L. A.** (1999). Linguistic styles: language use as an individual difference. *Journal of Personality and Social Psychology*, 77(6).
- Pennington, J., Socher, R., and Manning, C. D.** (2014). Glove: Global vectors for word representation. In *EMNLP 2014*, 1532–1543.
- Percival, W. K.** (1976). On the historical source of immediate constituent analysis. In McCawley, J. D. (Ed.), *Syntax and Semantics Volume 7, Notes from the Linguistic Under-ground*, 229–242. Academic Press.
- Perrault, C. R. and Allen, J.** (1980). A plan-based analysis of indirect speech acts. *American Journal of Computational Linguistics*, 6(3-4), 167–182.
- Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L.** (2018). Deep contextualized word representations. In *NAACL HLT 2018*, 2227–2237.

- Peterson**, G. E. and Barney, H. L. (1952). Control methods used in a study of the vowels. *JASA*, 24, 175–184.
- Petrov**, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *COLING/ACL 2006*, 433–440.
- Petrov**, S., Das, D., and McDonald, R. (2012). A universal part-of-speech tagset. In *LREC-12*.
- Petrov**, S. and McDonald, R. (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)*, Vol. 59.
- Phillips**, A. V. (1960). A question-answering routine. Tech. rep. 16, MIT AI Lab.
- Picard**, R. W. (1995). Affective computing. Tech. rep. 321, MIT Media Lab Perceptual Computing Technical Report. Revised November 26, 1995.
- Pieraccini**, R., Levin, E., and Lee, C.-H. (1991). Stochastic representation of conceptual structure in the ATIS task. In *Proceedings DARPA Speech and Natural Language Workshop*, 121–124.
- Pierrehumbert**, J. B. (1980). *The Phonology and Phonetics of English Intonation*. Ph.D. thesis, MIT.
- Pilehvar**, M. T. and Camacho-Collados, J. (2019). WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. In *NAACL HLT 2019*, 1267–1273.
- Pilehvar**, M. T., Jurgens, D., and Navigli, R. (2013). Align, disambiguate and walk: A unified approach for measuring semantic similarity. In *ACL 2013*, 1341–1351.
- Pitler**, E., Louis, A., and Nenkova, A. (2009). Automatic sense prediction for implicit discourse relations in text. In *ACL IJCNLP 2009*, 683–691.
- Pitler**, E. and Nenkova, A. (2009). Using syntax to disambiguate explicit discourse connectives in text. In *ACL IJCNLP 2009*, 13–16.
- Pitrelli**, J. F., Beckman, M. E., and Hirschberg, J. (1994). Evaluation of prosodic transcription labeling reliability in the ToBI framework. In *ICSLP-94*, Vol. 1, 123–126.
- Pitt**, M. A., Dilley, L., Johnson, K., Kiesling, S., Raymond, W. D., Hume, E., and Fosler-Lussier, E. (2007). Buckeye corpus of conversational speech (2nd release).. Department of Psychology, Ohio State University (Distributor).
- Pitt**, M. A., Johnson, K., Hume, E., Kiesling, S., and Raymond, W. D. (2005). The buckeye corpus of conversational speech: Labeling conventions and a test of transcriber reliability. *Speech Communication*, 45, 90–95.
- Plutchik**, R. (1962). *The emotions: Facts, theories, and a new model*. Random House.
- Plutchik**, R. (1980). A general psychoevolutionary theory of emotion. In Plutchik, R. and Kellerman, H. (Eds.), *Emotion: Theory, Research, and Experience, Volume 1*, 3–33. Academic Press.
- Poesio**, M., Stevenson, R., Di Eugenio, B., and Hitzeman, J. (2004). Centering: A parametric theory and its instantiations. *Computational Linguistics*, 30(3), 309–363.
- Poesio**, M., Stuckardt, R., and Versley, Y. (2016). *Anaphora resolution: Algorithms, resources, and applications*. Springer.
- Poesio**, M., Sturt, P., Artstein, R., and Filik, R. (2006). Underspecification and anaphora: Theoretical issues and preliminary evidence. *Discourse processes*, 42(2), 157–175.
- Poesio**, M. and Vieira, R. (1998). A corpus-based investigation of definite description use. *Computational Linguistics*, 24(2), 183–216.
- Polanyi**, L. (1988). A formal model of the structure of discourse. *Journal of Pragmatics*, 12.
- Polanyi**, L., Culy, C., van den Berg, M., Thione, G. L., and Ahn, D. (2004a). A rule based approach to discourse parsing. In *Proceedings of SIGDIAL*.
- Polanyi**, L., Culy, C., van den Berg, M., Thione, G. L., and Ahn, D. (2004b). Sentential structure and discourse parsing. In *ACL04 Discourse Annotation Workshop*.
- Polifroni**, J., Hirschman, L., Seneff, S., and Zue, V. W. (1992). Experiments in evaluating interactive spoken language systems. In *Proceedings DARPA Speech and Natural Language Workshop*, 28–33.
- Pollard**, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press.
- Ponzetto**, S. P. and Navigli, R. (2010). Knowledge-rich word sense disambiguation rivaling supervised systems. In *ACL 2010*, 1522–1531.
- Ponzetto**, S. P. and Strube, M. (2006). Exploiting semantic role labeling, WordNet and Wikipedia for coreference resolution. In *HLT-NAACL-06*, 192–199.
- Ponzetto**, S. P. and Strube, M. (2007). Knowledge derived from Wikipedia for computing semantic relatedness. *JAIR*, 30, 181–212.
- Porter**, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3), 130–137.
- Potts**, C. (2011). On the negativity of negation. In Li, N. and Lutz, D. (Eds.), *Proceedings of Semantics and Linguistic Theory 20*, 636–659. CLC Publications, Ithaca, NY.
- Pradhan**, S., Hovy, E. H., Marcus, M. P., Palmer, M., Ramshaw, L., and Weischedel, R. (2007a). OntoNotes: A unified relational semantic representation. In *Proceedings of ICSC*, 517–526.
- Pradhan**, S., Hovy, E. H., Marcus, M. P., Palmer, M., Ramshaw, L. A., and Weischedel, R. M. (2007b). Ontonotes: a unified relational semantic representation. *Int. J. Semantic Computing*, 1(4), 405–419.
- Pradhan**, S., Luo, X., Recasens, M., Hovy, E. H., Ng, V., and Strube, M. (2014). Scoring coreference partitions of predicted mentions: A reference implementation. In *ACL 2014*.
- Pradhan**, S., Moschitti, A., Xue, N., Ng, H. T., Björkelund, A., Uryupina, O., Zhang, Y., and Zhong, Z. (2013). Towards robust linguistic analysis using OntoNotes. In *CoNLL-13*, 143–152.
- Pradhan**, S., Moschitti, A., Xue, N., Uryupina, O., and Zhang, Y. (2012a). CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *CoNLL-12*.
- Pradhan**, S., Moschitti, A., Xue, N., Uryupina, O., and Zhang, Y. (2012b). Conll-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *CoNLL-12*.
- Pradhan**, S., Ramshaw, L., Marcus, M. P., Palmer, M., Weischedel, R., and Xue, N. (2011). CoNLL-2011 shared task: Modeling unrestricted coreference in OntoNotes. In *CoNLL-11*.
- Pradhan**, S., Ramshaw, L., Weischedel, R., MacBride, J., and Micciulla, L. (2007). Unrestricted coreference: Identifying entities and events in OntoNotes. In *Proceedings of ICSC 2007*, 446–453.
- Pradhan**, S., Ward, W., Hacioglu, K., Martin, J. H., and Jurafsky, D. (2005). Semantic role labeling using different syntactic views. In *ACL-05*.
- Prasad**, R., Dinesh, N., Lee, A., Miltakaki, E., Robaldo, L., Joshi, A. K., and Webber, B. L. (2008). The Penn Discourse TreeBank 2.0. In *LREC-08*.
- Prasad**, R., Webber, B. L., and Joshi, A. (2014). Reflections on the Penn Discourse Treebank, comparable corpora, and complementary annotation. *Computational Linguistics*, 40(4), 921–950.

- Price**, P. J., Ostendorf, M., Shattuck-Hufnagel, S., and Fong, C. (1991). The use of prosody in syntactic disambiguation. *JASA*, 90(6).
- Prince**, E. (1981a). Toward a taxonomy of given-new information. In Cole, P. (Ed.), *Radical Pragmatics*, 223–256. Academic Press.
- Prince**, E. (1981b). Toward a taxonomy of given-new information. In Cole, P. (Ed.), *Radical Pragmatics*, 223–255. Academic Press.
- Propp**, V. (1968). *Morphology of the Folktale* (2nd Ed.). University of Texas Press. Original Russian 1928. Translated by Laurence Scott.
- Pu**, X., Pappas, N., Henderson, J., and Popescu-Belis, A. (2018). Integrating weakly supervised word sense disambiguation into neural machine translation. *TACL*, 6, 635–649.
- Pullum**, G. K. and Ladusaw, W. A. (1996). *Phonetic Symbol Guide* (2nd Ed.). University of Chicago.
- Purver**, M. (2004). *The theory and use of clarification requests in dialogue*. Ph.D. thesis, University of London.
- Pustejovsky**, J. (1991). The generative lexicon. *Computational Linguistics*, 17(4).
- Pustejovsky**, J. (1995). *The Generative Lexicon*. MIT Press.
- Pustejovsky**, J. and Boguraev, B. K. (Eds.). (1996). *Lexical Semantics: The Problem of Polysemy*. Oxford University Press.
- Pustejovsky**, J., Castaño, J., Ingria, R., Saurí, R., Gaizauskas, R., Setzer, A., and Katz, G. (2003a). TimeML: robust specification of event and temporal expressions in text. In *Proceedings of the 5th International Workshop on Computational Semantics (IWCS-5)*.
- Pustejovsky**, J., Hanks, P., Saurí, R., See, A., Gaizauskas, R., Setzer, A., Radev, D., Sundheim, B., Day, D. S., Ferro, L., and Lazo, M. (2003b). The TIMEBANK corpus. In *Proceedings of Corpus Linguistics 2003 Conference*, 647–656. UCREL Technical Paper number 16.
- Pustejovsky**, J., Ingria, R., Saurí, R., Castaño, J., Littman, J., Gaizauskas, R., Setzer, A., Katz, G., and Mani, I. (2005). *The Specification Language TimeML*, chap. 27. Oxford.
- Qin**, L., Zhang, Z., and Zhao, H. (2016). A stacking gated neural architecture for implicit discourse relation classification. In *EMNLP 2016*, 2263–2270.
- Qin**, L., Zhang, Z., Zhao, H., Hu, Z., and Xing, E. (2017). Adversarial connective-exploiting networks for implicit discourse relation classification. In *ACL 2017*, 1006–1017.
- Quillian**, M. R. (1968). Semantic memory. In Minsky, M. (Ed.), *Semantic Information Processing*, 227–270. MIT Press.
- Quillian**, M. R. (1969). The teachable language comprehender: A simulation program and theory of language. *CACM*, 12(8), 459–476.
- Quirk**, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman.
- Rabiner**, L. R. and Schafer, R. W. (1978). *Digital Processing of Speech Signals*. Prentice Hall.
- Rabiner**, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.
- Rabiner**, L. R. and Juang, B. H. (1993). *Fundamentals of Speech Recognition*. Prentice Hall.
- Radford**, A. (1997). *Syntactic Theory and the Structure of English: A Minimalist Approach*. Cambridge University Press.
- Raganato**, A., Bovi, C. D., and Navigli, R. (2017a). Neural sequence learning models for word sense disambiguation. In *EMNLP 2017*, 1156–1167.
- Raganato**, A., Camacho-Collados, J., and Navigli, R. (2017b). Word sense disambiguation: A unified evaluation framework and empirical comparison. In *EACL-17*, 99–110.
- Raghunathan**, K., Lee, H., Rangarajan, S., Chambers, N., Surdeanu, M., Jurafsky, D., and Manning, C. D. (2010). A multi-pass sieve for coreference resolution. In *Proceedings of EMNLP 2010*, 492–501.
- Rahman**, A. and Ng, V. (2009). Supervised models for coreference resolution. In *EMNLP-09*, 968–977.
- Rahman**, A. and Ng, V. (2012). Resolving complex cases of definite pronouns: the Winograd Schema challenge. In *EMNLP 2012*, 777–789.
- Rajpurkar**, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. In *ACL 2018*.
- Rajpurkar**, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *EMNLP 2016*.
- Ramshaw**, L. A. and Marcus, M. P. (1995). Text chunking using transformation-based learning. In *Proceedings of the 3rd Annual Workshop on Very Large Corpora*, 82–94.
- Ranganath**, R., Jurafsky, D., and McFarland, D. A. (2013). Detecting friendly, flirtatious, awkward, and assertive speech in speed-dates. *Computer Speech and Language*, 27(1), 89–115.
- Raphael**, B. (1968). SIR: A computer program for semantic information retrieval. In Minsky, M. (Ed.), *Semantic Information Processing*, 33–145. MIT Press.
- Rashkin**, H., Bell, E., Choi, Y., and Volkova, S. (2017). Multilingual connotation frames: A case study on social media for targeted sentiment analysis and forecast. In *ACL 2017*, 459–464.
- Rashkin**, H., Singh, S., and Choi, Y. (2016). Connotation frames: A data-driven investigation. In *ACL 2016*, 311–321.
- Rashkin**, H., Smith, E. M., Li, M., and Boureau, Y.-L. (2019). Towards empathetic open-domain conversation models: A new benchmark and dataset. In *ACL 2019*, 5370–5381.
- Ratinov**, L. and Roth, D. (2012). Learning-based multi-sieve coreference resolution with knowledge. In *EMNLP 2012*, 1234–1244.
- Ratnaparkhi**, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP 1996*, 133–142.
- Ratnaparkhi**, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *EMNLP 1997*, 1–10.
- Ratnaparkhi**, A., Reynar, J. C., and Roukos, S. (1994). A maximum entropy model for prepositional phrase attachment. In *ARPA Human Language Technologies Workshop*, 250–255.
- Recasens**, M. and Hovy, E. H. (2011). BLANC: Implementing the Rand index for coreference evaluation. *Natural Language Engineering*, 17(4), 485–510.
- Recasens**, M., Hovy, E. H., and Martí, M. A. (2011). Identity, non-identity, and near-identity: Addressing the complexity of coreference. *Lingua*, 121(6), 1138–1152.
- Recasens**, M. and Martí, M. A. (2010). AnCorA-CO: Coreferentially annotated corpora for Spanish and Catalan. *Language Resources and Evaluation*, 44(4), 315–345.
- Reed**, C., Mochales Palau, R., Rowe, G., and Moens, M.-F. (2008). Language resources for studying argumentation. In *LREC-08*, 91–100.
- Rehder**, B., Schreiner, M. E., Wolfe, M. B. W., Laham, D., Landauer, T. K., and Kintsch, W. (1998). Using Latent Semantic Analysis to assess knowledge: Some technical considerations. *Discourse Processes*, 25(2-3), 337–354.
- Reichenbach**, H. (1947). *Elements of Symbolic Logic*. Macmillan, New York.

- Reichert**, T. A., Cohen, D. N., and Wong, A. K. C. (1973). An application of information theory to genetic mutations and the matching of polypeptide sequences. *Journal of Theoretical Biology*, 42, 245–261.
- Reichman**, R. (1985). *Getting Computers to Talk Like You and Me*. MIT Press.
- Resnik**, P. (1992). Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *COLING-92*, 418–424.
- Resnik**, P. (1993). Semantic classes and syntactic ambiguity. In *Proceedings of the workshop on Human Language Technology*, 278–283.
- Resnik**, P. (1996). Selectional constraints: An information-theoretic model and its computational realization. *Cognition*, 61, 127–159.
- Richardson**, M., Burges, C. J. C., and Renshaw, E. (2013). MCTest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP 2013*, 193–203.
- Riedel**, S., Yao, L., and McCallum, A. (2010). Modeling relations and their mentions without labeled text. In *Machine Learning and Knowledge Discovery in Databases*, 148–163. Springer.
- Riedel**, S., Yao, L., McCallum, A., and Marlin, B. M. (2013). Relation extraction with matrix factorization and universal schemas. In *NAACL HLT 2013*.
- Riesbeck**, C. K. (1975). Conceptual analysis. In Schank, R. C. (Ed.), *Conceptual Information Processing*, 83–156. American Elsevier, New York.
- Riloff**, E. (1993). Automatically constructing a dictionary for information extraction tasks. In *AAAI-93*, 811–816.
- Riloff**, E. (1996). Automatically generating extraction patterns from untagged text. In *AAAI-96*, 117–124.
- Riloff**, E. and Jones, R. (1999). Learning dictionaries for information extraction by multi-level bootstrapping. In *AAAI-99*, 474–479.
- Riloff**, E. and Schmelzenbach, M. (1998). An empirical approach to conceptual case frame acquisition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, 49–56.
- Riloff**, E. and Shepherd, J. (1997). A corpus-based approach for building semantic lexicons. In *EMNLP 1997*.
- Riloff**, E. and Thelen, M. (2000). A rule-based question answering system for reading comprehension tests. In *Proceedings of ANLP/NAACL workshop on reading comprehension tests*, 13–19.
- Riloff**, E. and Wiebe, J. (2003). Learning extraction patterns for subjective expressions. In *EMNLP 2003*.
- Ritter**, A., Cherry, C., and Dolan, B. (2010). Unsupervised modeling of twitter conversations. In *HLT-NAACL*.
- Ritter**, A., Cherry, C., and Dolan, B. (2011). Data-driven response generation in social media. In *EMNLP-11*, 583–593.
- Ritter**, A., Etzioni, O., and Mausam (2010). A latent dirichlet allocation method for selectional preferences. In *ACL 2010*, 424–434.
- Ritter**, A., Zettlemoyer, L., Mausam, and Etzioni, O. (2013). Modeling missing data in distant supervision for information extraction.. *TACL*, 1, 367–378.
- Robins**, R. H. (1967). *A Short History of Linguistics*. Indiana University Press, Bloomington.
- Rohde**, D. L. T., Gonnerman, L. M., and Plaut, D. C. (2006). An improved model of semantic similarity based on lexical co-occurrence. *CACM*, 8, 627–633.
- Rooth**, M., Riezler, S., Prescher, D., Carroll, G., and Beil, F. (1999). Inducing a semantically annotated lexicon via EM-based clustering. In *ACL-99*, 104–111.
- Rosenblatt**, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386–408.
- Rosenfeld**, R. (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10, 187–228.
- Rosenthal**, S. and McKeown, K. (2017). Detecting influencers in multiple online genres. *ACM Transactions on Internet Technology (TOIT)*, 17(2).
- Rothe**, S., Ebert, S., and Schütze, H. (2016). Ultradense Word Embeddings by Orthogonal Transformation. In *NAACL HLT 2016*.
- Roy**, N., Pineau, J., and Thrun, S. (2000). Spoken dialog management for robots. In *ACL-00*.
- Rudinger**, R., Naradowsky, J., Leonard, B., and Van Durme, B. (2018). Gender bias in coreference resolution. In *NAACL HLT 2018*.
- Rumelhart**, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, Vol. 2, 318–362. MIT Press.
- Rumelhart**, D. E. and McClelland, J. L. (1986a). On learning the past tense of English verbs. In Rumelhart, D. E. and McClelland, J. L. (Eds.), *Parallel Distributed Processing*, Vol. 2, 216–271. MIT Press.
- Rumelhart**, D. E. and McClelland, J. L. (1986b). *Parallel Distributed Processing*. MIT Press.
- Rumelhart**, D. E., McClelland, J. L., and The PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. Bradford Books (MIT Press).
- Ruppenhofer**, J., Ellsworth, M., Petrucc, M. R. L., Johnson, C. R., Baker, C. F., and Scheffczyk, J. (2016). FrameNet II: Extended theory and practice..
- Ruppenhofer**, J., Sporleder, C., Morante, R., Baker, C. F., and Palmer, M. (2010). SemEval-2010 task 10: Linking events and their participants in discourse. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, 45–50.
- Russell**, J. A. (1980). A circumplex model of affect. *Journal of personality and social psychology*, 39(6), 1161–1178.
- Russell**, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach* (2nd Ed.). Prentice Hall.
- Rutherford**, A. and Xue, N. (2015). Improving the inference of implicit discourse relations via classifying explicit discourse connectives. In *NAACL HLT 2015*, 799–808.
- Sacks**, H., Schegloff, E. A., and Jefferson, G. (1974). A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4), 696–735.
- Sag**, I. A. and Liberman, M. Y. (1975). The intonational disambiguation of indirect speech acts. In *CLS-75*, 487–498. University of Chicago.
- Sag**, I. A., Wasow, T., and Bender, E. M. (Eds.). (2003). *Syntactic Theory: A Formal Introduction*. CSLI Publications, Stanford, CA.
- Sagae**, K. (2009). Analysis of discourse structure with syntactic dependencies and data-driven shift-reduce parsing. In *IWPT-09*, 81–84.
- Sahami**, M., Dumais, S. T., Heckerman, D., and Horvitz, E. (1998). A Bayesian approach to filtering junk e-mail. In *AAAI Workshop on Learning for Text Categorization*, 98–105.
- Sakoe**, H. and Chiba, S. (1971). A dynamic programming approach to continuous speech recognition. In *Proceedings of the Seventh International Congress on Acoustics*, Vol. 3, 65–69. Akadémiai Kiadó.

- Salomaa, A.** (1969). Probabilistic and weighted grammars. *Information and Control*, 15, 529–544.
- Salton, G.** (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice Hall.
- Sampson, G.** (1987). Alternative grammatical coding systems. In Garfield, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, 165–183. Longman.
- Samuelsson, C.** (1993). Morphological tagging based entirely on Bayesian inference. In *9th Nordic Conference on Computational Linguistics NODALIDA-93*. Stockholm.
- Sankoff, D.** (1972). Matching sequences under deletion-insertion constraints. *Proceedings of the National Academy of Sciences*, 69, 4–6.
- Sankoff, D.** and Labov, W. (1979). On the uses of variable rules. *Language in society*, 8(2-3), 189–222.
- Sap, M.**, Prasetio, M. C., Holtzman, A., Rashkin, H., and Choi, Y. (2017). Connotation frames of power and agency in modern films. In *EMNLP 2017*, 2329–2334.
- Scha, R.** and Polanyi, L. (1988). An augmented context free grammar for discourse. In *COLING-88*, 573–577.
- Schabes, Y.** (1990). *Mathematical and Computational Aspects of Lexicalized Grammars*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.
- Schabes, Y.** (1992). Stochastic lexicalized tree-adjoining grammars. In *COLING-92*, 426–433.
- Schabes, Y.**, Abeillé, A., and Joshi, A. K. (1988). Parsing strategies with ‘lexicalized’ grammars: Applications to Tree Adjoining Grammars. In *COLING-88*, 578–583.
- Schank, R. C.** (1972). Conceptual dependency: A theory of natural language processing. *Cognitive Psychology*, 3, 552–631.
- Schank, R. C.** and Abelson, R. P. (1975). Scripts, plans, and knowledge. In *Proceedings of IJCAI-75*, 151–157.
- Schank, R. C.** and Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum.
- Schegloff, E. A.** (1968). Sequencing in conversational openings. *American Anthropologist*, 70, 1075–1095.
- Scherer, K. R.** (2000). Psychological models of emotion. In Borod, J. C. (Ed.), *The neuropsychology of emotion*, 137–162. Oxford.
- Schiebinger, L.** (2019). Machine translation: Analyzing gender. <http://genderedinnovations.stanford.edu/case-studies/nlp.html#tabs-2>.
- Schneider, N.**, Hwang, J. D., Srikumar, V., Prange, J., Blodgett, A., Moeller, S. R., Stern, A., Bitan, A., and Abend, O. (2018). Comprehensive supersense disambiguation of English prepositions and possessives. In *ACL 2018*, 185–196.
- Schone, P.** and Jurafsky, D. (2000). Knowledge-free induction of morphology using latent semantic analysis. In *CoNLL-00*.
- Schone, P.** and Jurafsky, D. (2001). Knowledge-free induction of inflectional morphologies. In *NAACL 2001*.
- Schönfinkel, M.** (1924). Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92, 305–316. English translation appears in *From Frege to Gödel: A Source Book in Mathematical Logic*, Harvard University Press, 1967.
- Schuster, M.** and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45, 2673–2681.
- Schütze, H.** (1992a). Context space. In Goldman, R. (Ed.), *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*.
- Schütze, H.** (1992b). Dimensions of meaning. In *Proceedings of Supercomputing '92*, 787–796. IEEE Press.
- Schütze, H.** (1997a). *Ambiguity Resolution in Language Learning – Computational and Cognitive Models*. CSLI, Stanford, CA.
- Schütze, H.** (1997b). *Ambiguity Resolution in Language Learning: Computational and Cognitive Models*. CSLI Publications, Stanford, CA.
- Schütze, H.** (1998). Automatic word sense discrimination. *Computational Linguistics*, 24(1), 97–124.
- Schütze, H.**, Hull, D. A., and Pedersen, J. (1995). A comparison of classifiers and document representations for the routing problem. In *SIGIR-95*, 229–237.
- Schütze, H.** and Pedersen, J. (1993). A vector model for syntagmatic and paradigmatic relatedness. In *Proceedings of the 9th Annual Conference of the UW Centre for the New OED and Text Research*, 104–113.
- Schütze, H.** and Singer, Y. (1994). Part-of-speech tagging using a variable memory Markov model. In *ACL-94*, 181–187.
- Schwartz, H. A.**, Eichstaedt, J. C., Kern, M. L., Dziurzynski, L., Ramones, S. M., Agrawal, M., Shah, A., Kosinski, M., Stillwell, D., Seligman, M. E. P., and Ungar, L. H. (2013). Personality, gender, and age in the language of social media: The open-vocabulary approach. *PloS one*, 8(9), e73791.
- Schwenk, H.** (2007). Continuous space language models. *Computer Speech & Language*, 21(3), 492–518.
- Séaghdha, D. O.** (2010). Latent variable models of selectional preference. In *ACL 2010*, 435–444.
- Seddah, D.**, Tsarfaty, R., Kübler, S., Candito, M., Choi, J. D., Farkas, R., Foster, J., Goenaga, I., Gojenola, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przeździecki, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Wolinski, M., Wróblewska, A., and Villemonte de la Clérgerie, E. (2013). Overview of the SPMRL 2013 shared task: cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the 4th Workshop on Statistical Parsing of Morphologically-Rich Languages*.
- Sekine, S.** and Collins, M. (1997). The evalb software. <http://cs.nyu.edu/cs/projects/proteus/evalb>.
- Seneff, S.** and Zue, V. W. (1988). Transcription and alignment of the TIMIT database. In *Proceedings of the Second Symposium on Advanced Man-Machine Interface through Spoken Language*.
- Senrich, R.**, Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *ACL 2016*.
- Seo, M.**, Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2017). Bidirectional attention flow for machine comprehension. In *ICLR 2017*.
- Serban, I. V.**, Lowe, R., Henderson, P., Charlin, L., and Pineau, J. (2018). A survey of available corpora for building data-driven dialogue systems: The journal version. *Dialogue & Discourse*, 9(1), 1–49.
- Sgall, P.**, Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in its Pragmatic Aspects*. Reidel.
- Shang, L.**, Lu, Z., and Li, H. (2015). Neural responding machine for short-text conversation. In *ACL 2015*, 1577–1586.
- Shannon, C. E.** (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379–423. Continued in the following volume.
- Shannon, C. E.** (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, 30, 50–64.
- Sheil, B. A.** (1976). Observations on context free parsing. *SMIL: Statistical Methods in Linguistics*, 1, 71–109.
- Shockley, L.** (2003). *Sound Patterns of Spoken English*. Blackwell.

- Shoup, J. E.** (1980). Phonological aspects of speech recognition. In Lea, W. A. (Ed.), *Trends in Speech Recognition*, 125–138. Prentice Hall.
- Shriberg, E.**, Bates, R., Taylor, P., Stolcke, A., Jurafsky, D., Ries, K., Coccaro, N., Martin, R., Meteer, M., and Van Ess-Dykema, C. (1998). Can prosody aid the automatic classification of dialog acts in conversational speech?. *Language and Speech (Special Issue on Prosody and Conversation)*, 41(3-4), 439–487.
- Sidner, C. L.** (1979). Towards a computational theory of definite anaphora comprehension in English discourse. Tech. rep. 537, MIT Artificial Intelligence Laboratory, Cambridge, MA.
- Sidner, C. L.** (1983). Focusing in the comprehension of definite anaphora. In Brady, M. and Berwick, R. C. (Eds.), *Computational Models of Discourse*, 267–330. MIT Press.
- Silverman, K.**, Beckman, M. E., Pitrelli, J. F., Ostendorf, M., Wightman, C. W., Price, P. J., Pierrehumbert, J. B., and Hirschberg, J. (1992). ToBI: A standard for labelling English prosody. In *IICSLP-92*, Vol. 2, 867–870.
- Simmons, R. F.** (1965). Answering English questions by computer: A survey. *CACM*, 8(1), 53–70.
- Simmons, R. F.** (1973). Semantic networks: Their computation and use for understanding English sentences. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, 61–113. W.H. Freeman and Co.
- Simmons, R. F.**, Klein, S., and McConologue, K. (1964). Indexing and dependency logic for answering english questions. *American Documentation*, 15(3), 196–204.
- Simons, G. F.** and Fennig, C. D. (2018). Ethnologue: Languages of the world, twenty-first edition.. SIL International.
- Singh, S. P.**, Litman, D. J., Kearns, M., and Walker, M. A. (2002). Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *JAIR*, 16, 105–133.
- Sleator, D.** and Temperley, D. (1993). Parsing English with a link grammar. In *IWPT-93*.
- Small, S. L.** and Rieger, C. (1982). Parsing and comprehending with Word Experts. In Lehnert, W. G. and Ringle, M. H. (Eds.), *Strategies for Natural Language Processing*, 89–147. Lawrence Erlbaum.
- Smith, V. L.** and Clark, H. H. (1993). On the course of answering questions. *Journal of Memory and Language*, 32, 25–38.
- Smolensky, P.** (1988). On the proper treatment of connectionism. *Behavioral and brain sciences*, 11(1), 1–23.
- Smolensky, P.** (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2), 159–216.
- Snow, R.**, Jurafsky, D., and Ng, A. Y. (2005). Learning syntactic patterns for automatic hypernym discovery. In Saul, L. K., Weiss, Y., and Bottou, L. (Eds.), *NIPS 17*, 1297–1304. MIT Press.
- Snow, R.**, Prakash, S., Jurafsky, D., and Ng, A. Y. (2007). Learning to merge word senses. In *EMNLP/CoNLL 2007*, 1005–1014.
- Snyder, B.** and Palmer, M. (2004). The English all-words task. In *SENSEVAL-3*, 41–43.
- Socher, R.**, Huval, B., Manning, C. D., and Ng, A. Y. (2012). Semantic compositionality through recursive matrix-vector spaces. In *EMNLP 2012*, 1201–1211.
- Soderland, S.**, Fisher, D., Aseltine, J., and Lehnert, W. G. (1995). CRYSTAL: Inducing a conceptual dictionary. In *IJCAI-95*, 1134–1142.
- Søgaard, A.** (2010). Simple semi-supervised training of part-of-speech taggers. In *ACL 2010*, 205–208.
- Søgaard, A.** and Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. In *ACL 2016*.
- Søgaard, A.**, Johannsen, A., Plank, B., Hovy, D., and Alonso, H. M. (2014). What's in a p-value in NLP?. In *CoNLL-14*.
- Solorio, T.**, Blair, E., Maharjan, S., Bethard, S., Diab, M., Ghoneim, M., Hawwari, A., AlGhamdi, F., Hirschberg, J., Chang, A., and Fung, P. (2014). Overview for the first shared task on language identification in code-switched data. In *Proceedings of the First Workshop on Computational Approaches to Code Switching*, 62–72.
- Somasundaran, S.**, Burstein, J., and Chodorow, M. (2014). Lexical chaining for measuring discourse coherence quality in test-taker essays. In *COLING-14*, 950–961.
- Soon, W. M.**, Ng, H. T., and Lim, D. C. Y. (2001). A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4), 521–544.
- Sordoni, A.**, Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses. In *NAACL HLT 2015*, 196–205.
- Soricut, R.** and Marcu, D. (2003). Sentence level discourse parsing using syntactic and lexical information. In *HLT-NAACL-03*, 149–156.
- Soricut, R.** and Marcu, D. (2006). Discourse generation using utility-trained coherence models. In *COLING/ACL 2006*, 803–810.
- Sparck Jones, K.** (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1), 11–21.
- Sparck Jones, K.** (1986). *Synonymy and Semantic Classification*. Edinburgh University Press, Edinburgh. Republication of 1964 PhD Thesis.
- Spitkovsky, V. I.** and Chang, A. X. (2012). A cross-lingual dictionary for English Wikipedia concepts. In *LREC-12*.
- Sporleder, C.** and Lascaris, A. (2005). Exploiting linguistic cues to classify rhetorical relations. In *RANLP-05*.
- Sporleder, C.** and Lapata, M. (2005). Discourse chunking and its application to sentence compression. In *HLT-EMNLP-05*, 257–264.
- Srivastava, N.**, Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2014). Dropout: a simple way to prevent neural networks from overfitting.. *JMLR*, 15(1), 1929–1958.
- Stab, C.** and Gurevych, I. (2014a). Annotating argument components and relations in persuasive essays. In *COLING-14*, 1501–1510.
- Stab, C.** and Gurevych, I. (2014b). Identifying argumentative discourse structures in persuasive essays. In *EMNLP 2014*, 46–56.
- Stab, C.** and Gurevych, I. (2017). Parsing argumentation structures in persuasive essays. *Computational Linguistics*, 43(3), 619–659.
- Stalnaker, R. C.** (1978). Assertion. In Cole, P. (Ed.), *Pragmatics: Syntax and Semantics Volume 9*, 315–332. Academic Press.
- Stamatatos, E.** (2009). A survey of modern authorship attribution methods. *JASIST*, 60(3), 538–556.
- Steðe, M.** (2011). *Discourse processing*. Morgan & Claypool.
- Steðe, M.** and Schneider, J. (2018). *Argumentation Mining*. Morgan & Claypool.
- Steðe, M.** (1989). Constituency and coordination in a combinatorial grammar. In Baltin, M. R. and Kroch, A. S. (Eds.), *Alternative Conceptions of Phrase Structure*, 201–231. University of Chicago.
- Steðe, M.** (1996). *Surface Structure and Interpretation*. MIT Press. Linguistic Inquiry Monograph, 30.

- Steedman, M.** (2000). *The Syntactic Process*. The MIT Press.
- Steedman, M.** (2007). Information-structural semantics for English intonation. In Lee, C., Gordon, M., and Büring, D. (Eds.), *Topic and Focus: Cross-Linguistic Perspectives on Meaning and Intonation*, 245–264. Springer.
- Stetina, J.** and Nagao, M. (1997). Corpus based PP attachment ambiguity resolution with a semantic dictionary. In Zhou, J. and Church, K. W. (Eds.), *Proceedings of the Fifth Workshop on Very Large Corpora*, 66–80.
- Stevens, K. N.** (1998). *Acoustic Phonetics*. MIT Press.
- Stevens, K. N.** and House, A. S. (1955). Development of a quantitative description of vowel articulation. *JASA*, 27, 484–493.
- Stevens, K. N.** and House, A. S. (1961). An acoustical theory of vowel production and some of its implications. *Journal of Speech and Hearing Research*, 4, 303–320.
- Stevens, K. N.**, Kasowski, S., and Fant, G. M. (1953). An electrical analog of the vocal tract. *JASA*, 25(4), 734–742.
- Stevens, S. S.** and Volkmann, J. (1940). The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, 53(3), 329–353.
- Stevens, S. S.**, Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *JASA*, 8, 185–190.
- Stiefelman, L. J.**, Arons, B., Schmandt, C., and Hulteen, E. A. (1993). VoiceNotes: A speech interface for a hand-held voice notetaker. In *Human Factors in Computing Systems: INTERCHI '93 Conference Proceedings*, 179–186.
- Stolcke, A.** (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2), 165–202.
- Stolcke, A.** (1998). Entropy-based pruning of backoff language models. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 270–274.
- Stolcke, A.** (2002). SRILM – an extensible language modeling toolkit. In *ICSLP-02*.
- Stolcke, A.**, Ries, K., Coccaro, N., Shriberg, E., Bates, R., Jurafsky, D., Taylor, P., Martin, R., Meteer, M., and Van Ess-Dykema, C. (2000). Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3), 339–371.
- Stoltz, W. S.**, Tannenbaum, P. H., and Carstensen, F. V. (1965). A stochastic approach to the grammatical coding of English. *CACM*, 8(6), 399–405.
- Stone, P.**, Dunphy, D., Smith, M., and Ogilvie, D. (1966). *The General Inquirer: A Computer Approach to Content Analysis*. Cambridge, MA: MIT Press.
- Stoyanov, S.** and Johnston, M. (2015). Localized error detection for targeted clarification in a virtual assistant. In *ICASSP-15*, 5241–5245.
- Stoyanov, S.**, Liu, A., and Hirschberg, J. (2013). Modelling human clarification strategies. In *SIGDIAL 2013*, 137–141.
- Stoyanov, S.**, Liu, A., and Hirschberg, J. (2014). Towards natural clarification questions in dialogue systems. In *AISB symposium on questions, discourse and dialogue*.
- Strötgen, J.** and Gertz, M. (2013). Multilingual and cross-domain temporal tagging. *Language Resources and Evaluation*, 47(2), 269–298.
- Strube, M.** and Hahn, U. (1996). Functional centering. In *ACL-96*, 270–277.
- Subba, R.** and Di Eugenio, B. (2009). An effective discourse parser that uses rich linguistic information. In *NAACL HLT 2009*, 566–574.
- Suendermann, D.**, Evanini, K., Liscombe, J., Hunter, P., Dayanidhi, K., and Pieraccini, R. (2009). From rule-based to statistical grammars: Continuous improvement of large-scale spoken dialog systems. In *ICASSP-09*, 4713–4716.
- Sundheim, B.** (Ed.). (1991). *Proceedings of MUC-3*.
- Sundheim, B.** (Ed.). (1992). *Proceedings of MUC-4*.
- Sundheim, B.** (Ed.). (1993). *Proceedings of MUC-5*, Baltimore, MD.
- Sundheim, B.** (Ed.). (1995). *Proceedings of MUC-6*.
- Surdeanu, M.** (2013). Overview of the TAC2013 Knowledge Base Population evaluation: English slot filling and temporal slot filling. In *TAC-13*.
- Surdeanu, M.**, Harabagiu, S., Williams, J., and Aarseth, P. (2003). Using predicate-argument structures for information extraction. In *ACL-03*, 8–15.
- Surdeanu, M.**, Hicks, T., and Valenzuela-Escarcega, M. A. (2015). Two practical rhetorical structure theory parsers. In *NAACL HLT 2015*, 1–5.
- Surdeanu, M.**, Johansson, R., Meyers, A., Márquez, L., and Nivre, J. (2008a). The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL-08*, 159–177.
- Surdeanu, M.**, Johansson, R., Meyers, A., Márquez, L., and Nivre, J. (2008b). The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL-08*, 159–177.
- Sutskever, I.**, Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 3104–3112.
- Sweet, H.** (1877). *A Handbook of Phonetics*. Clarendon Press.
- Swerts, M.**, Litman, D. J., and Hirschberg, J. (2000). Corrections in spoken dialogue systems. In *ICSLP-00*.
- Swier, R.** and Stevenson, S. (2004). Unsupervised semantic role labelling. In *EMNLP 2004*, 95–102.
- Switzer, P.** (1965). Vector images in document retrieval. In Stevens, M. E., Giuliano, V. E., and Heilprin, L. B. (Eds.), *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings*. Washington, D.C., USA, March 17, 1964, 163–171. <https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbmsmiscellaneouspub269.pdf>.
- Talbot, D.** and Osborne, M. (2007). Smoothed Bloom Filter Language Models: Tera-Scale LMs on the Cheap. In *EMNLP/CoNLL 2007*, 468–476.
- Talmor, A.** and Berant, J. (2018). The web as a knowledge-base for answering complex questions. In *NAACL HLT 2018*.
- Tan, C.**, Niculae, V., Danescu-Niculescu-Mizil, C., and Lee, L. (2016). Winning arguments: Interaction dynamics and persuasion strategies in good-faith online discussions. In *WWW-16*, 613–624.
- Tannen, D.** (1979). What's in a frame? Surface evidence for underlying expectations. In Freedle, R. (Ed.), *New Directions in Discourse Processing*, 137–181. Ablex.
- ter Meulen, A.** (1995). *Representing Time in Natural Language*. MIT Press.
- Tesnière, L.** (1959). *Éléments de Syntaxe Structurale*. Librairie C. Klincksieck, Paris.
- Tetreault, J. R.** (2001). A corpus-based evaluation of centering and pronoun resolution. *Computational Linguistics*, 27(4), 507–520.

- Teufel**, S., Carletta, J., and Moens, M. (1999). An annotation scheme for discourse-level argumentation in research articles. In *EACL-99*.
- Teufel**, S., Siddharthan, A., and Batchelor, C. (2009). Towards domain-independent argumentative zoning: Evidence from chemistry and computational linguistics. In *EMNLP-09*, 1493–1502.
- Thede**, S. M. and Harper, M. P. (1999). A second-order hidden Markov model for part-of-speech tagging. In *ACL-99*, 175–182.
- Thompson**, K. (1968). Regular expression search algorithm. *CACM*, 11(6), 419–422.
- Tibshirani**, R. J. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1), 267–288.
- Titov**, I. and Khoddam, E. (2014). Unsupervised induction of semantic roles within a reconstruction-error minimization framework. In *NAACL HLT 2015*.
- Titov**, I. and Klementiev, A. (2012). A Bayesian approach to unsupervised semantic role induction. In *EACL-12*, 12–22.
- Tomkins**, S. S. (1962). *Affect, imagery, consciousness: Vol. I. The positive affects*. Springer.
- Toutanova**, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL-03*.
- Trichelaire**, P., Emami, A., Cheung, J. C. K., Trischler, A., Suleman, K., and Diaz, F. (2018). On the evaluation of common-sense reasoning in natural language understanding. In *NeurIPS 2018 Workshop on Critiquing and Correcting Trends in Machine Learning*.
- Trnka**, K., Yarrington, D., McCaw, J., McCoy, K. F., and Pennington, C. (2007). The effects of word prediction on communication rate for AAC. In *NAACL-HLT 07*, 173–176.
- Tseng**, H., Jurafsky, D., and Manning, C. D. (2005). Morphological features help POS tagging of unknown words across language varieties. In *Proceedings of the 4th SIGHAN Workshop on Chinese Language Processing*.
- Tsvetkov**, Y., Schneider, N., Hovy, D., Bhatia, A., Faruqui, M., and Dyer, C. (2014). Augmenting English adjective senses with supersenses. In *LREC-14*.
- Tur**, G. and De Mori, R. (2011). *Spoken language understanding: Systems for extracting semantic information from speech*. John Wiley & Sons.
- Turian**, J., Ratinov, L., and Bengio, Y. (2010). Word representations: a simple and general method for semi-supervised learning. In *ACL 2010*, 384–394.
- Turney**, P. D. (2002). Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In *ACL-02*.
- Turney**, P. D. and Littman, M. (2003). Measuring praise and criticism: Inference of semantic orientation from association. *ACM Transactions on Information Systems (TOIS)*, 21, 315–346.
- Uryupina**, O., Artstein, R., Bristot, A., Cavicchio, F., Delogu, F., Rodriguez, K. J., and Poesio, M. (2019). Annotating a broad range of anaphoric phenomena, in a variety of genres: The ARRAU corpus. *Natural Language Engineering*, 1–34.
- UzZaman**, N., Llorens, H., Derczynski, L., Allen, J., Verhagen, M., and Pustejovsky, J. (2013). Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In *SemEval-13*, 1–9.
- van Benthem**, J. and ter Meulen, A. (Eds.). (1997). *Handbook of Logic and Language*. MIT Press.
- van Deemter**, K. and Kibble, R. (2000). On coreferring: coreference in MUC and related annotation schemes. *Computational Linguistics*, 26(4), 629–637.
- van der Maaten**, L. and Hinton, G. E. (2008). Visualizing high-dimensional data using t-sne. *JMLR*, 9, 2579–2605.
- van Rijsbergen**, C. J. (1975). *Information Retrieval*. Butterworths.
- Van Valin, Jr.**, R. D. and La Polla, R. (1997). *Syntax: Structure, Meaning, and Function*. Cambridge University Press.
- Velikovich**, L., Blair-Goldensohn, S., Hannan, K., and McDonald, R. (2010). The viability of web-derived polarity lexicons. In *NAACL HLT 2010*, 777–785.
- Venditti**, J. J. (2005). The j-tobi model of Japanese intonation. In Jun, S.-A. (Ed.), *Prosodic Typology and Transcription: A Unified Approach*. Oxford University Press.
- Vendler**, Z. (1967). *Linguistics in Philosophy*. Cornell University Press, Ithaca, NY.
- Verhagen**, M., Gaizauskas, R., Schilder, F., Hepple, M., Moszkowicz, J., and Pustejovsky, J. (2009). The tempeval challenge: identifying temporal relations in text. *Language Resources and Evaluation*, 43(2), 161–179.
- Verhagen**, M., Mani, I., Sauri, R., Knippen, R., Jang, S. B., Littman, J., Rumshisky, A., Phillips, J., and Pustejovsky, J. (2005). Automating temporal annotation with tarsqi. In *ACL-05*, 81–84.
- Versley**, Y. (2008). Vagueness and referential ambiguity in a large-scale annotated corpus. *Research on Language and Computation*, 6(3–4), 333–353.
- Vieira**, R. and Poesio, M. (2000). An empirically based system for processing definite descriptions. *Computational Linguistics*, 26(4), 539–593.
- Vilain**, M., Burger, J. D., Aberdeen, J., Connolly, D., and Hirschman, L. (1995). A model-theoretic coreference scoring scheme. In *MUC-6*.
- Vintsyuk**, T. K. (1968). Speech discrimination by dynamic programming. *Cybernetics*, 4(1), 52–57. Russian Kibernetika 4(1):81–88. 1968.
- Vinyals**, O. and Le, Q. V. (2015). A neural conversational model. In *Proceedings of ICML Deep Learning Workshop*.
- Viterbi**, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2), 260–269.
- Vossen**, P., Görög, A., Laan, F., Van Gompel, M., Izquierdo, R., and Van Den Bosch, A. (2011). Dutch-semcor: building a semantically annotated corpus for dutch. In *Proceedings of eLex*, 286–296.
- Voutilainen**, A. (1995). Morphological disambiguation. In Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.), *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, 165–284. Mouton de Gruyter.
- Voutilainen**, A. (1999). Handcrafted rules. In van Halteren, H. (Ed.), *Syntactic Wordclass Tagging*, 217–246. Kluwer.
- Vrandečić**, D. and Krötzsch, M. (2014). Wikidata: a free collaborative knowledge base. *CACM*, 57(10), 78–85.
- Wade**, E., Shriberg, E., and Price, P. J. (1992). User behaviors affecting speech recognition. In *ICSLP-92*, 995–998.
- Wagner**, R. A. and Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21, 168–173.
- Walker**, M. A. (2000). An application of reinforcement learning to dialogue strategy selection in a spoken dialogue system for email. *JAIR*, 12, 387–416.

- Walker, M. A., Fromer, J. C., and Narayanan, S. S.** (1998). Learning optimal dialogue strategies: A case study of a spoken dialogue agent for email. In *COLING/ACL-98*, 1345–1351.
- Walker, M. A., Iida, M., and Cote, S.** (1994). Japanese discourse and the process of centering. *Computational Linguistics*, 20(2), 193–232.
- Walker, M. A., Joshi, A. K., and Prince, E. (Eds.)** (1998). *Centering in Discourse*. Oxford University Press.
- Walker, M. A., Kamm, C. A., and Litman, D. J.** (2001). Towards developing general models of usability with PARADISE. *Natural Language Engineering: Special Issue on Best Practice in Spoken Dialogue Systems*, 6(3), 363–377.
- Walker, M. A. and Whittaker, S.** (1990). Mixed initiative in dialogue: An investigation into discourse segmentation. In *ACL-90*, 70–78.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R.** (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR 2017*.
- Wang, H., Lu, Z., Li, H., and Chen, E.** (2013). A dataset for research on short-text conversations.. In *EMNLP 2013*, 935–945.
- Wang, S. and Manning, C. D.** (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *ACL 2012*, 90–94.
- Wang, Y., Li, S., and Yang, J.** (2018). Toward fast and accurate neural discourse segmentation. In *EMNLP 2018*.
- Ward, W. and Issar, S.** (1994). Recent improvements in the CMU spoken language understanding system. In *ARPA Human Language Technologies Workshop*.
- Weaver, W.** (1949/1955). Translation. In Locke, W. N. and Boothe, A. D. (Eds.), *Machine Translation of Languages*, 15–23. MIT Press. Reprinted from a memorandum written by Weaver in 1949.
- Webber, B. L.** (1978). *A Formal Approach to Discourse Anaphora*. Ph.D. thesis, Harvard University.
- Webber, B. L.** (1983). So what can we talk about now?. In Brady, M. and Berwick, R. C. (Eds.), *Computational Models of Discourse*, 331–371. The MIT Press. Reprinted in Grosz et al. (1986).
- Webber, B. L.** (1991). Structure and ostension in the interpretation of discourse deixis. *Language and Cognitive Processes*, 6(2), 107–135.
- Webber, B. L. and Baldwin, B.** (1992). Accommodating context change. In *ACL-92*, 96–103.
- Webber, B. L., Egg, M., and Kordonis, V.** (2012). Discourse structure and language technology. *Natural Language Engineering*, 18(4), 437–490.
- Webber, B. L.** (1988). Discourse deixis: Reference to discourse segments. In *ACL-88*, 113–122.
- Webster, K., Recasens, M., Axelrod, V., and Baldridge, J.** (2018). Mind the gap: A balanced corpus of gendered ambiguous pronouns. *TACL*, 6, 605–617.
- Weinschenk, S. and Barker, D. T.** (2000). *Designing Effective Speech Interfaces*. Wiley.
- Weischedel, R., Hovy, E. H., Marcus, M. P., Palmer, M., Belvin, R., Pradhan, S., Ramshaw, L. A., and Xue, N.** (2011). Ontonotes: A large training corpus for enhanced processing. In Joseph Olive, Caitlin Christianson, J. M. (Ed.), *Handbook of Natural Language Processing and Machine Translation: DARPA Global Automatic Language Exploitation*, 54–63. Springer.
- Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L. A., and Palmucci, J.** (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2), 359–382.
- Weizenbaum, J.** (1966). ELIZA – A computer program for the study of natural language communication between man and machine. *CACM*, 9(1), 36–45.
- Weizenbaum, J.** (1976). *Computer Power and Human Reason: From Judgement to Calculation*. W.H. Freeman and Company.
- Wells, J. C.** (1982). *Accents of English*. Cambridge University Press.
- Wen, T.-H., Gašić, M., Kim, D., Mrkšić, N., Su, P.-H., Vandyke, D., and Young, S. J.** (2015a). Stochastic language generation in dialogue using recurrent neural networks with convolutional sentence reranking. In *SIGDIAL 2015*, 275–284.
- Wen, T.-H., Gašić, M., Mrkšić, N., Su, P.-H., Vandyke, D., and Young, S. J.** (2015b). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP 2015*.
- Werbos, P.** (1974). *Beyond regression : new tools for prediction and analysis in the behavioral sciences /*. Ph.D. thesis, Harvard University.
- Werbos, P. J.** (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10), 1550–1560.
- Widrow, B. and Hoff, M. E.** (1960). Adaptive switching circuits. In *IRE WESCON Convention Record*, Vol. 4, 96–104.
- Wiebe, J.** (1994). Tracking point of view in narrative. *Computational Linguistics*, 20(2), 233–287.
- Wiebe, J.** (2000). Learning subjective adjectives from corpora. In *AAAI-00*, 735–740.
- Wiebe, J., Bruce, R. F., and O'Hara, T. P.** (1999). Development and use of a gold-standard data set for subjectivity classifications. In *ACL-99*, 246–253.
- Wierzbicka, A.** (1992). *Semantics, Culture, and Cognition: University Human Concepts in Culture-Specific Configurations*. Oxford University Press.
- Wierzbicka, A.** (1996). *Semantics: Primes and Universals*. Oxford University Press.
- Wilensky, R.** (1983). *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley.
- Wilks, Y.** (1973). An artificial intelligence approach to machine translation. In Schank, R. C. and Colby, K. M. (Eds.), *Computer Models of Thought and Language*, 114–151. W.H. Freeman.
- Wilks, Y.** (1975a). An intelligent analyzer and understander of English. *CACM*, 18(5), 264–274.
- Wilks, Y.** (1975b). Preference semantics. In Keenan, E. L. (Ed.), *The Formal Semantics of Natural Language*, 329–350. Cambridge Univ. Press.
- Wilks, Y.** (1975c). A preferential, pattern-seeking, semantics for natural language inference. *Artificial Intelligence*, 6(1), 53–74.
- Williams, J. D., Raux, A., and Henderson, M.** (2016). The dialog state tracking challenge series: A review. *Dialogue & Discourse*, 7(3), 4–33.
- Williams, J. D. and Young, S. J.** (2007). Partially observable markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(1), 393–422.
- Wilson, T., Wiebe, J., and Hoffmann, P.** (2005). Recognizing contextual polarity in phrase-level sentiment analysis. In *HLT-EMNLP-05*, 347–354.
- Winograd, T.** (1972). *Understanding Natural Language*. Academic Press.
- Winston, P. H.** (1977). *Artificial Intelligence*. Addison Wesley.
- Wiseman, S., Rush, A. M., and Shieber, S. M.** (2016). Learning global features for coreference resolution. In *NAACL HLT 2016*.

- Wiseman**, S., Rush, A. M., Shieber, S. M., and Weston, J. (2015). Learning anaphoricity and antecedent ranking features for coreference resolution. In *ACL 2015*, 1416–1426.
- Witten**, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4), 1085–1094.
- Witten**, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques* (2nd Ed.). Morgan Kaufmann.
- Wittgenstein**, L. (1953). *Philosophical Investigations*. (Translated by Anscombe, G.E.M.). Blackwell.
- Wolf**, F. and Gibson, E. (2005). Representing discourse coherence: A corpus-based analysis. *Computational Linguistics*, 31(2), 249–287.
- Woods**, W. A. (1967). *Semantics for a Question-Answering System*. Ph.D. thesis, Harvard University.
- Woods**, W. A. (1973). Progress in natural language understanding. In *Proceedings of AFIPS National Conference*, 441–450.
- Woods**, W. A. (1975). What's in a link: Foundations for semantic networks. In Bobrow, D. G. and Collins, A. M. (Eds.), *Representation and Understanding: Studies in Cognitive Science*, 35–82. Academic Press.
- Woods**, W. A. (1978). Semantics and quantification in natural language question answering. In Yovits, M. (Ed.), *Advances in Computers*, 2–64. Academic.
- Woods**, W. A., Kaplan, R. M., and Nash-Webber, B. L. (1972). The lunar sciences natural language information system: Final report. Tech. rep. 2378, BBN.
- Woodsend**, K. and Lapata, M. (2015). Distributed representations for unsupervised semantic role labeling. In *EMNLP 2015*, 2482–2491.
- Wu**, F. and Weld, D. S. (2007). Autonomously semantifying Wikipedia. In *CIKM-07*, 41–50.
- Wu**, F. and Weld, D. S. (2010). Open information extraction using Wikipedia. In *ACL 2010*, 118–127.
- Wu**, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G. S., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.
- Wundt**, W. (1900). *Völkerpsychologie: eine Untersuchung der Entwicklungsgesetze von Sprache, Mythos, und Sitte*. W. Engelmann, Leipzig. Band II: Die Sprache, Zweiter Teil.
- Xia**, F. and Palmer, M. (2001). Converting dependency structures to phrase structures. In *HLT-01*, 1–5.
- Xu**, P., Saghaf, H., Kang, J. S., Long, T., Bose, A. J., Cao, Y., and Cheung, J. C. K. (2019). A cross-domain transferable neural coherence model. In *ACL 2019*, 678–687.
- Xue**, N., Ng, H. T., Pradhan, S., Rutherford, A., Webber, B. L., Wang, C., and Wang, H. (2016). CoNLL 2016 shared task on multilingual shallow discourse parsing. In *Proceedings of the CoNLL-16 shared task*.
- Xue**, N. and Palmer, M. (2004). Calibrating features for semantic role labeling. In *EMNLP 2004*.
- Yamada**, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In Noord, G. V. (Ed.), *IWPT-03*, 195–206.
- Yan**, Z., Duan, N., Bao, J.-W., Chen, P., Zhou, M., Li, Z., and Zhou, J. (2016). DocChat: An information retrieval approach for chatbot engines using unstructured documents. In *ACL 2016*.
- Yang**, D., Chen, J., Yang, Z., Jurafsky, D., and Hovy, E. H. (2019). Let's make your request more persuasive: Modeling persuasive strategies via semi-supervised neural nets on crowdfunding platforms. In *NAACL HLT 2019*, 3620–3630.
- Yang**, X., Zhou, G., Su, J., and Tan, C. L. (2003). Coreference resolution using competition learning approach. In *ACL-03*, 176–183.
- Yang**, Y., Yih, W.-t., and Meek, C. (2015). Wikiqa: A challenge dataset for open-domain question answering. In *EMNLP 2015*.
- Yang**, Y. and Pedersen, J. (1997). A comparative study on feature selection in text categorization. In *ICML*, 412–420.
- Yankelovich**, N., Levow, G.-A., and Marx, M. (1995). Designing SpeechActs: Issues in speech user interfaces. In *Human Factors in Computing Systems: CHI '95 Conference Proceedings*, 369–376.
- Yarowsky**, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *ACL-95*, 189–196.
- Yasseri**, T., Kornai, A., and Kertész, J. (2012). A practical approach to language complexity: a Wikipedia case study. *PloS one*, 7(11).
- Yih**, W.-t., Richardson, M., Meek, C., Chang, M.-W., and Suh, J. (2016). The value of semantic parse labeling for knowledge base question answering. In *ACL 2016*, 201–206.
- Yngve**, V. H. (1955). Syntax and the problem of multiple meaning. In Locke, W. N. and Booth, A. D. (Eds.), *Machine Translation of Languages*, 208–226. MIT Press.
- Young**, S. J., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K. (2010). The Hidden Information State model: A practical framework for POMDP-based spoken dialogue management. *Computer Speech & Language*, 24(2), 150–174.
- Younger**, D. H. (1967). Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10, 189–208.
- Yu**, M. and Dredze, M. (2014). Improving lexical embeddings with semantic knowledge. In *ACL 2014*, 545–550.
- Yu**, N., Zhang, M., and Fu, G. (2018). Transition-based neural RST parsing with implicit syntax features. In *COLING-18*, 559–570.
- Yu**, Y., Zhu, Y., Liu, Y., Liu, Y., Peng, S., Gong, M., and Zeldes, A. (2019). GumDrop at the DISRPT2019 shared task: A model stacking approach to discourse unit segmentation and connective detection. In *Proceedings of the Workshop on Discourse Relation Parsing and Treebanking 2019*.
- Zapirain**, B., Agirre, E., Márquez, L., and Surdeanu, M. (2013). Selectional preferences for semantic role classification. *Computational Linguistics*, 39(3), 631–663.
- Zavrel**, J. and Daelemans, W. (1997). Memory-based learning: Using similarity for smoothing. In *ACL/EACL-97*, 436–443.
- Zelle**, J. M. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *AAAI-96*, 1050–1055.
- Zeman**, D. (2008). Reusable tagset conversion using tagset drivers.. In *LREC-08*.
- Zeman**, D., Popel, M., Straka, M., Hajíč, J., Nivre, J., Ginter, F., Luoto-lahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F. M., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinková, S., Hajic, Jr., J., Hlaváčová, J., Kettnerová, V., Uresová, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C. D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H.,

- de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., de Paiva, V., Droganova, K., Alonso, H. M., Çöltekin, Ç., Sulubacak, U., Uszkorait, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., El-Kahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonça, G., Lando, T., Nitisoroj, R., and Li, J. (2017). Conll 2017 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, August 3-4, 2017*, 1–19.
- Zettlemoyer**, L. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence, UAI'05*, 658–666.
- Zhang**, R., dos Santos, C. N., Yasunaga, M., Xiang, B., and Radev, D. (2018). Neural coreference resolution with deep biaffine attention by joint mention detection and mention clustering. In *ACL 2018*, 102–107.
- Zhang**, Y. and Clark, S. (2008). A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *EMNLP-08*, 562–571.
- Zhang**, Y. and Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. In *ACL 2011*, 188–193.
- Zhao**, H., Chen, W., Kit, C., and Zhou, G. (2009). Multilingual dependency learning: A huge feature engineering method to semantic dependency parsing. In *CoNLL-09*, 55–60.
- Zhao**, J., Wang, T., Yatskar, M., Cotterell, R., Ordonez, V., and Chang, K.-W. (2019). Gender bias in contextualized word embeddings. In *NAACL HLT 2019*, 629–634.
- Zhao**, J., Wang, T., Yatskar, M., Ordonez, V., and Chang, K.-W. (2017). Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In *EMNLP 2017*.
- Zhao**, J., Wang, T., Yatskar, M., Ordonez, V., and Chang, K.-W. (2018a). Gender bias in coreference resolution: Evaluation and debiasing methods. In *NAACL HLT 2018*.
- Zhao**, J., Zhou, Y., Li, Z., Wang, W., and Chang, K.-W. (2018b). Learning gender-neutral word embeddings. In *EMNLP 2018*, 4847–4853.
- Zheng**, J., Vilnis, L., Singh, S., Choi, J. D., and McCallum, A. (2013). Dynamic knowledge-base alignment for coreference resolution. In *CoNLL-13*, 153–162.
- Zhong**, Z. and Ng, H. T. (2010). It makes sense: A wide-coverage word sense disambiguation system for free text. In *ACL 2010*, 78–83.
- Zhou**, D., Bousquet, O., Lal, T. N., Weston, J., and Schölkopf, B. (2004). Learning with local and global consistency. In *NIPS 2004*.
- Zhou**, G., Su, J., Zhang, J., and Zhang, M. (2005). Exploring various knowledge in relation extraction. In *ACL-05*, 427–434.
- Zhou**, J. and Xu, W. (2015a). End-to-end learning of semantic role labeling using recurrent neural networks. In *ACL 2015*, 1127–1137.
- Zhou**, J. and Xu, W. (2015b). End-to-end learning of semantic role labeling using recurrent neural networks. In *ACL 2015*, 1127–1137.
- Zhou**, L., Gao, J., Li, D., and Shum, H.-Y. (2018). The design and implementation of XiaoIce, an empathetic social chatbot.
- Zhou**, L., Ticrea, M., and Hovy, E. H. (2004). Multi-document biography summarization. In *EMNLP 2004*.
- Zhou**, Y. and Xue, N. (2015). The Chinese Discourse TreeBank: a Chinese corpus annotated with discourse relations. *Language Resources and Evaluation*, 49(2), 397–431.
- Zhu**, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. Tech. rep. CMU-CALD-02, CMU.
- Zhu**, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003*, 912–919.
- Zue**, V. W., Glass, J., Goodine, D., Leung, H., Phillips, M., Polifroni, J., and Seneff, S. (1989). Preliminary evaluation of the VOYAGER spoken language system. In *Proceedings DARPA Speech and Natural Language Workshop*, 160–167.

Author Index

- Ševčíková, M., 277, 296
 Štěpánek, J., 277, 296
- Aarseth, P., 392
 Abadi, M., 137
 Abeillé, A., 271, 272
 Abelson, R. P., 349, 379,
 391, 392
 Abend, O., 360
 Aberdeen, J., 435, 436, 441
 Abney, S. P., 166, 244, 268,
 270, 272, 527
 Aceró, A., 544
 Agarwal, A., 137
 Agarwal, O., 436
 Aggarwal, C. C., 73
 Aggarwal, S., 506, 516
 Agichtein, E., 338, 352
 Agirre, E., 119, 371, 393
 Agrawal, M., 409–411
 Ahn, D., 28, 462
 Ahn, Y.-Y., 400
 Aho, A. V., 244, 278
 Ajdukiewicz, K., 223
 Alcalde, H. F., 294
 AlGhamdi, F., 13
 Algoet, P. H., 51
 Allen, J., 323, 347, 348,
 353, 490, 516
 Alonso, H. M., 73, 294
 Althoff, T., 460
 Altun, Y., 360
 Amir, S., 190
 Amsler, R. A., 371
 An, J., 400
 Androutsopoulos, I., 72
 Angelard-Gontier, N., 498,
 514
 Antiga, L., 137
 Anttila, A., 167, 296
 Aone, C., 441
 Appelt, D. E., 350–352, 441
 Appling, S., 514
 Ariel, M., 421
 Arons, B., 513
 Artstein, R., 425, 442, 498
 Aseltine, J., 352
 Asher, N., 451, 462, 463
 Auer, S., 334, 476
 Auli, M., 497
 Austerweil, J., 462
 Austin, J. L., 489, 515
 Awadallah, A. H., 506
 Axelrod, V., 438, 439
 Ayers, G. M., 528
- Ba, J., 137
 Baayen, R. H., 28, 539
 Babko-Malaya, O., 371
 Baccianella, S., 403
 Bach, K., 489
 Backus, J. W., 205, 229
 Badmaeva, E., 294
- Bagga, A., 435
 Bahdanau, D., 185
 Bahl, L. R., 53, 166
 Baker, C. F., 379, 392
 Baker, J. K., 53, 253, 271
 Baldridge, J., 427, 429,
 430, 436, 438, 439,
 441, 462
 Baldwin, B., 421, 435
 Baldwin, T., 65, 73
 Ballard, D. H., 142
 Ballesteros, M., 180, 331,
 352
 Balogh, J., 502, 508, 509,
 513
 Bamman, D., 459
 Banaji, M. R., 118
 Banarescu, L., 298
 Banea, C., 119
 Banerjee, E., 294
 Bangalore, S., 272
 Banko, M., 352, 469, 472
 Bansal, M., 336, 352
 Bao, J.-W., 496
 Bar-Hillel, Y., 223
 Barham, P., 137
 Barker, C., 442
 Barker, D. T., 508
 Barney, H. L., 543
 Barrett, L., 258, 271
 Barzilay, R., 444, 452,
 454–457, 462, 463
 Basile, P., 366
 Batchelor, C., 461
 Bates, R., 516
 Bauer, F. L., 229
 Bauer, J., 22
 Baum, L. E., 558, 563
 Baum, L. F., 513
 Baumer, G., 73
 Bayes, T., 58
 Bazell, C. E., 229
 Bean, D., 419, 427, 441
 Bear, J., 350–352
 Becker, C., 334, 476
 Beckman, M. E., 528
 Bedi, G., 445
 Beil, F., 393
 Bejček, E., 277, 296
 Bell, E., 392, 412, 413
 Bell, T. C., 54
 Bellegarda, J. R., 121, 515
 Bellman, R., 24, 28
 Belvin, R., 277
 Bender, E. M., 230
 Bengio, S., 498
 Bengio, Y., 54, 94, 111,
 121, 127–129, 138,
 142, 185, 498
 Bengtson, E., 431
 Bennett, E., 451, 463
 Bennett, S. W., 441
 Benuš, Š., 516
- Berant, J., 478, 479, 483,
 485
 Berg-Kirkpatrick, T., 70,
 71, 73
 Berger, A., 93
 Bergsma, S., 393, 427, 441
 Bernstein, M. S., 415
 Bethard, S., 13, 22, 353,
 392, 393
 Bhat, I., 288
 Bhat, R. A., 288
 Bhatia, A., 360
 Biber, D., 230
 Bies, A., 217, 219
 Bikell, D. M., 352
 Bills, S., 339, 352
 Biran, O., 452
 Birch, A., 18, 20
 Bird, S., 16, 17, 28
 Bisani, M., 73
 Bishop, C. M., 73, 93
 Bitan, A., 360
 Bizer, C., 334, 476
 Björkelund, A., 384, 430
 Black, A. W., 190, 439
 Black, E., 268, 270, 271,
 371
 Blair, E., 13
 Blair-Goldensohn, S., 414
 Blei, D. M., 121
 Blodgett, A., 360
 Blodgett, S. L., 13, 40, 65
 Bloomfield, L., 220, 229,
 373
 Blunsom, P., 483, 485
 Bobrow, D. G., 322, 391,
 498, 499, 515
 Bobrow, R. J., 515
 Bod, R., 271
 Boersma, P., 534, 542, 544
 Bogomolov, M., 73
 Boguraev, B. K., 371, 441,
 485
 Bohus, D., 509
 Bojanowski, P., 121
 Bolinger, D., 526
 Bollacker, K., 334, 476
 Bolukbasi, T., 117, 118,
 439
 Bonial, C., 298
 Booth, R. J., 64, 398, 399
 Booth, T. L., 246, 247, 271
 Bordes, A., 468, 473, 474,
 485
 Borges, J. L., 56
 Bose, A. J., 457, 458, 462
 Boser, B., 142
 Boser, B. E., 142
 Bottou, L., 111, 121, 190,
 352, 392
 Bougares, F., 185
 Boureau, Y.-L., 516
 Bourlard, H., 142
 Bousquet, O., 414
- Bovi, C. D., 371
 Bowman, S. R., 438, 498
 Boyd-Graber, J., 485
 Brachman, R. J., 322
 Brants, T., 49, 158, 159,
 166, 167
 Braud, C., 447, 449, 452,
 462
 Bréal, M., 96
 Breck, E., 472, 485
 Brennan, S. E., 441, 453
 Bresnan, J., 223, 229, 246
 Brevdo, E., 137
 Brill, E., 272, 469, 472
 Brin, S., 352
 Briscoe, T., 270, 272, 371
 Bristot, A., 425
 Broadhead, M., 352
 Brockett, C., 497
 Brockman, W., 117
 Brockmann, C., 388
 Broschart, J., 144
 Bruce, B. C., 516
 Bruce, R., 371
 Bruce, R. F., 394
 Brysbaert, M., 398
 Bryson, J. J., 118, 439
 Buchholz, S., 296
 Buck, C., 41
 Budanitsky, A., 96
 Budzianowski, P., 510
 Buechel, S., 400
 Bullinaria, J. A., 121
 Bulyko, I., 506
 Bunke, H., 190
 Bunker, R. T., 362
 Burchardt, A., 294
 Burger, J. D., 435, 436,
 441, 472, 485
 Burges, C. J. C., 484, 496
 Burget, L., 121, 176
 Burkett, D., 70, 71, 73
 Burnett, D., 513
 Burstein, J., 445
 Byrne, W., 542
- Cafarella, M., 352
 Cai, S., 298
 Caliskan, A., 118, 439
 Camacho-Collados, J., 119,
 365, 367
 Camburu, O.-M., 438
 Campbell, J., 31
 Candito, M., 297
 Cao, J., 507
 Cao, Y., 19, 457, 458, 462
 Caputo, A., 366
 Carberry, S., 490
 Cardie, C., 119, 190, 352,
 426, 431, 441, 460
 Carenini, G., 462
 Carletta, J., 421, 461
 Carlson, L., 445, 447, 462
 Carmel, D., 485

- Carpenter, B., 511
 Carpenter, R., 492
 Carreras, X., 384, 392
 Carrillo, F., 445
 Carroll, G., 393
 Carroll, J., 270, 272
 Carstensen, F. V., 166
 Casale, J., 507
 Casanueva, I., 510
 Castaño, J., 343, 345, 347
 Cavicchio, F., 425
 Cecchi, G. A., 445
 Cer, D., 119
 Černocký, J., 176
 Černocký, J. H., 121
 Chafe, W. L., 421, 452, 463
 Chahuneau, V., 407, 408
 Chai, J. Y., 392
 Chambers, N., 350, 353, 389, 426, 442
 Chanan, G., 137
 Chaney, A., 296
 Chang, A., 13, 426, 442
 Chang, A. X., 345, 353, 437, 478, 482
 Chang, B., 366, 462
 Chang, J. S., 371
 Chang, K.-W., 117, 118, 427, 430, 438, 439
 Chang, M.-W., 19, 122, 475, 483
 Chaplot, D. S., 363
 Charles, W. G., 414
 Charlin, L., 495, 497, 498
 Charniak, E., 166, 220, 258, 271, 370, 462, 463
 Che, W., 392
 Chelba, C., 250
 Chen, B., 415
 Chen, C., 436
 Chen, D., 286, 468, 473, 474, 485
 Chen, E., 456, 496
 Chen, J., 460
 Chen, J. N., 371
 Chen, K., 111, 119, 121, 190
 Chen, M. Y., 414
 Chen, P., 496
 Chen, Q., 495
 Chen, S. F., 46, 48, 54
 Chen, W., 392
 Chen, X., 17, 99, 542
 Chen, Z., 19, 137
 Cherry, C., 495–497
 Cheung, J. C. K., 438
 Chiba, S., 563
 Chierchia, G., 323
 Chinchor, N., 73, 352
 Chintala, S., 137
 Chiticariu, L., 331
 Chiu, J. P. C., 190
 Cho, K., 185
 Chodorow, M., 445
 Choi, E., 483
 Choi, J. D., 288, 297, 437
 Choi, Y., 392, 393, 412, 413, 483
 Chomsky, C., 322, 476, 484
 Chomsky, N., 53, 205, 222, 229, 423
 Chou, A., 478, 479, 483
 Christodoulopoulos, C., 167
 Chu, Y.-J., 291
 Chu-Carroll, J., 485, 490, 511, 516
 Chung, T., 506, 516
 Church, A., 308
 Church, K. W., 14, 44, 46, 49, 54, 108, 109, 166, 244, 363, 388
 Cialdini, R. B., 460
 Ciaramita, M., 296, 360
 Cinková, S., 294
 Citro, C., 137
 Clark, C., 122, 363, 485
 Clark, E., 96
 Clark, H. H., 12, 421, 489, 516, 517
 Clark, J., 543
 Clark, J. H., 48, 49, 54
 Clark, K., 400–402, 430, 442
 Clark, P., 484
 Clark, S., 121, 167, 286
 CMU, 526, 540
 Coavoux, M., 447, 462
 Coccaro, N., 121, 516
 Cohen, D. N., 563
 Cohen, K. B., 325
 Cohen, M. H., 502, 508, 509, 513
 Cohen, P. R., 516
 Colaresi, M. P., 406–408
 Colby, K. M., 495, 515
 Cole, R. A., 513
 Coleman, J., 543
 Collins, M., 220–222, 256–259, 270, 271, 278, 478, 527
 Collobert, R., 111, 121, 190, 352, 392
 Colombe, J. B., 352
 Colosimo, M., 352
 Çöltekin, Ç., 294
 Connolly, D., 435, 436, 441
 Conrad, S., 230
 Conti, J., 513
 Cooper, F. S., 543
 Copelli, M., 445
 Copestate, A., 371
 Coppola, B., 485
 Corcoran, C. M., 445
 Corrado, G. S., 19, 111, 119, 121, 190
 Cote, S., 441
 Cotterell, R., 439
 Cottrell, G. W., 370
 Courville, A., 94, 127–129, 142
 Cover, T. M., 50, 51
 Covington, M., 280, 296
 Cowhey, I., 484
 Cox, D., 93
 Cox, S. J., 73
 Crammer, K., 296
 Craven, M., 352
 Cretu, A.-M., 438
 Cruse, D. A., 122
 Csomai, A., 418, 436, 437
 Cucerzan, S., 437
 Culicover, P. W., 230
 Culy, C., 462
 Curran, J. R., 167
 Cyganiak, R., 334, 476
 Daelemans, W., 272
 Dagan, I., 109, 110, 121, 363
 Dagan, O., 368
 Dai, A. M., 498
 Dale, R., 463
 Danescu-Niculescu-Mizil, C., 460
 Dang, H. T., 352, 371, 376, 392
 Danieli, M., 508, 512
 Danilevsky, M., 331
 Das, D., 296, 485
 Das, S. R., 414
 Davidson, D., 312, 322
 Davidson, T., 514
 Davies, M., 102, 117
 Davis, A., 137
 Davis, E., 323, 438
 Day, D. S., 347, 441
 Dayanidhi, K., 505
 de Courtenay, B., 543
 Dean, J., 19, 49, 111, 119, 121, 137, 190
 Deerwester, S. C., 120
 DeJong, G. F., 352, 392
 Delattre, P. C., 543
 Della Pietra, S. A., 93
 Della Pietra, V. J., 93
 Delogu, F., 425
 Demner-Fushman, D., 325
 Dempster, A. P., 558
 Denis, P., 427, 429, 430, 436, 441
 Denker, J. S., 142
 Derczynski, L., 353
 DeRose, S. J., 166
 Desmaison, A., 137
 Devin, M., 137
 DeVito, Z., 137
 Devlin, J., 19, 122, 475
 de Lacalle, O. L., 371
 de Marneffe, M.-C., 147, 165, 217, 275, 294, 296, 441
 De Mori, R., 516
 de Paiva, V., 294
 de Villiers, J. H., 513
 Di Eugenio, B., 441
 Diab, M., 13, 119, 371
 Diaz, F., 438
 Dienes, P., 164
 Digman, J. M., 409
 Dilley, L., 542
 Dinesh, N., 447
 Dingare, S., 421
 Ditman, T., 445
 Di Eugenio, B., 462, 463
 Dligach, D., 369
 Do, Q. N. T., 392, 393
 Doddington, G., 203
 Dodge, J., 368
 Dolan, B., 371, 495–497
 dos Santos, C. N., 336, 352, 430, 442
 Downey, D., 352
 Dowty, D. R., 322, 389
 Dozat, T., 275, 294, 296
 Dredze, M., 368
 Droganova, K., 294
 Dror, R., 73
 Duan, N., 496
 Duboué, P. A., 485
 Ducharme, R., 111, 121, 138
 Duda, R. O., 369
 Dumais, S. T., 64, 72, 118, 120, 121, 469, 472
 Dunn, H. K., 543
 Dunphy, D., 64, 397, 414
 Durrett, G., 430, 431, 437
 duVerle, D. A., 462
 Dyer, C., 180, 190, 331, 352, 360, 368, 483
 Dziurzynski, L., 409–411
 Eagon, J. A., 563
 Earley, J., 235, 244
 Ebert, S., 415
 Echihabi, A., 463
 Eckert, W., 516
 Edmonds, J., 291
 Edmonds, P., 371
 Efron, B., 70
 Egg, M., 463
 Egghe, L., 28
 Eichstaedt, J. C., 409–411
 Eisenstein, J., 449, 462
 Eisner, J., 296, 550
 Ejerhed, E. I., 244
 Ekman, P., 396
 El-Kahky, A., 294
 Elisseeff, A., 73
 Ellis, D., 542
 Ellsworth, M., 379, 380
 Elman, J. L., 142, 170, 171, 190
 Elsner, M., 462, 463
 Elvevåg, B., 445
 Emami, A., 438
 Erk, K., 393
 Eryigit, G., 296
 Espeholt, L., 485
 Esuli, A., 403
 Etzioni, O., 340, 341, 352, 393, 478, 479, 484
 Evanini, K., 505
 Evans, C., 334, 476
 Evans, N., 144
 Evert, S., 121
 Fader, A., 340, 341, 352, 478, 479
 Fan, J., 485
 Fano, R. M., 108

- Fant, G. M., 543
 Fandy, M., 513
 Farhadi, A., 485
 Farkas, R., 297
 Faruqui, M., 360, 368
 Fast, E., 415
 Fauconnier, G., 442
 Fazel-Zarandi, M., 507
 Feldman, J. A., 142
 Fellbaum, C., 359, 370, 371
 Feng, S., 485
 Feng, V. W., 445, 460, 462, 463
 Fennig, C. D., 13
 Fensel, D., 321
 Ferguson, J., 550
 Ferguson, M., 217, 219
 Fernandez, R., 190
 Fernandes, E. R., 430
 Fernández, S., 190
 Ferro, L., 343, 345, 347
 Ferrucci, D. A., 485
 Fessler, L., 514
 Field, A., 411
 Fikes, R. E., 516
 Filik, R., 442
 Fillmore, C. J., 229, 230, 322, 374, 379, 391
 Finegan, E., 230
 Finkel, J., 22
 Finkelstein, L., 118
 Finlayson, M. A., 459
 Firth, J. R., 94, 98, 515
 Fisch, A., 468, 473, 474, 485
 Fischer, M. J., 24, 563
 Fisher, D., 352
 Fitt, S., 541
 Flickinger, D., 268, 270
 Flynn, T. N., 399
 Fodor, J. A., 120, 322, 387
 Fodor, P., 485
 Fokoue-Nkoutche, A., 485
 Foland, W., 190
 Foland, Jr., W. R., 392
 Folds, D., 514
 Foltz, P. W., 445, 457
 Fong, C., 527
 Forchini, P., 495
 Forney, Jr., G. D., 563
 Fosler, E., 34
 Fosler-Lussier, E., 542
 Foster, J., 297
 Fox, B. A., 421
 Fox Tree, J. E., 12
 Francis, H. S., 254
 Francis, M. E., 64, 398, 399
 Francis, W. N., 11, 166
 Frank, E., 73, 93
 Franz, A., 49, 272
 Fraser, N. M., 513
 Freitag, D., 352
 Fried, G., 514
 Friedman, J. H., 73, 93
 Friedman, M. W., 441, 453
 Fromer, J. C., 508
 Frostig, R., 478, 479, 483
 Fry, D. B., 543
 Fu, G., 449, 450, 462
 Fung, P., 13, 516, 542
 Furnas, G. W., 120
 Fyshe, A., 122
 Gabow, H. N., 292
 Gabriel, R., 495
 Gabrilovich, E., 118
 Gage, P., 18
 Gaizauskas, R., 343, 345, 347
 Gale, W. A., 44, 46, 54, 363, 388
 Galil, Z., 292
 Galley, M., 497
 Gandhe, S., 498
 Gao, J., 49, 491, 492, 496, 497
 Gao, Q., 19
 Gao, S., 506, 516
 Gardner, M., 122, 363, 485
 Garg, N., 118, 439
 Garside, R., 166, 167
 Gašić, M., 503, 504, 510
 Gauvain, J.-L., 54, 121
 Gazdar, G., 216
 Gdaniec, C., 268, 270
 Georgescu, M., 298
 Georgila, K., 516
 Geramifard, A., 507
 Gerber, L., 343, 345
 Gerber, M., 392
 Gerbino, E., 508, 512
 Gerten, J., 498
 Gertz, M., 345, 353
 Ghahramani, Z., 414
 Ghemawat, S., 137
 Ghoneim, M., 13
 Giangola, J. P., 502, 508, 509, 513
 Gibson, E., 462
 Gil, D., 144
 Gilbert, G. N., 513
 Gildea, D., 190, 382, 392
 Giles, C. L., 190
 Gillick, L., 73
 Ginter, F., 147, 165, 217, 275, 294, 296
 Ginzburg, J., 511
 Girard, G., 96
 Giuliano, V. E., 120
 Givón, T., 254
 Glass, J., 512
 Glavaš, G., 368
 Glennie, A., 244
 Godfrey, J., 11, 203, 495
 Goebel, R., 393, 427
 Goenaga, I., 297
 Goffman, E., 391
 Gojenola, K., 297
 Gokirmak, M., 294
 Gold, B., 544
 Goldberg, J., 506
 Goldberg, T. E., 445
 Goldberg, Y., 110, 116, 118, 121, 122, 142, 147, 165, 190, 217, 275, 296, 297
 Goldberger, J., 363
 Goldwater, S., 167
 Gondek, D. C., 485
 Gonen, H., 118
 Gong, M., 452
 Gonnerman, L. M., 115
 Gonzalez-Agirre, A., 119
 Good, M. D., 513
 Goodfellow, I., 127–129, 137, 142
 Goodine, D., 512
 Goodman, J., 46, 48, 54
 Goodman, N., 451, 463
 Goodwin, C., 516
 Gopalakrishnan, K., 495
 Görög, A., 362
 Gosling, S. D., 409
 Gottardi, A., 495
 Gould, J. D., 512, 513
 Gould, S. J., 94
 Gouws, S., 19
 Gravano, A., 516
 Gravano, L., 338, 352
 Grave, E., 121
 Graves, A., 190
 Green, B. F., 322, 476, 484
 Green, J., 229
 Green, L., 13, 65
 Green, S., 297
 Greenbaum, S., 230
 Greenberg, S., 542
 Greene, B. B., 166
 Greenwald, A. G., 117, 118
 Grefenstette, E., 483, 485
 Gregory, M. L., 254
 Grenager, T., 392
 Grice, H. P., 491
 Griffitt, K., 298
 Grishman, R., 268, 270, 350, 352, 378, 417, 436
 Gross, S., 137
 Grosz, B. J., 421, 441, 444, 452, 453, 463, 516
 Grover, C., 329
 Gruber, J. S., 374, 391
 Guinaudeau, C., 463
 Gulcehre, C., 185
 Gulikers, L., 539
 Gundel, J. K., 421
 Guo, Y., 392
 Gurevych, I., 459, 460
 Gusfield, D., 27, 28
 Guyon, I., 73
 Habash, N., 294, 297
 Habernal, I., 459, 460
 Hacioglu, K., 392
 Haddow, B., 18, 20
 Haghhighi, A., 442
 Hahn, U., 400, 441
 Hajič, J., 147, 164, 165, 217, 270, 275, 277, 294, 296
 Hajíčová, E., 277, 295, 296
 Hajishirzi, H., 437, 442, 485
 Hakkani-Tür, D., 164, 495, 506, 510, 516
 Hall, J., 296
 Halliday, M. A. K., 444, 457
 Hamilton, W. L., 117, 400–402
 Han, Q., 17
 Hanks, P., 108, 109, 347
 Hannan, K., 414
 Hansen, B., 513
 Harabagiu, S., 392, 468
 Harnish, R., 489
 Harper, M. P., 167
 Harris, K., 294
 Harris, R. A., 513
 Harris, Z. S., 94, 98, 166, 244
 Harshman, R. A., 120
 Hart, P. E., 369
 Hart, T., 49
 Hasan, R., 444, 457
 Hastie, T., 73, 93
 Hathi, S., 73
 Hatzivassiloglou, V., 401, 402, 414
 Haverinen, K., 275, 296
 Haviland, S. E., 421
 Hawkins, J. A., 421
 Hawwari, A., 13
 He, H., 483
 He, L., 383, 384, 392, 427, 430–432, 434, 435, 442
 He, Z., 366
 Heafield, K., 41, 48, 49, 54
 Heaps, H. S., 12, 28
 Hearst, M. A., 334, 335, 340, 352, 371, 457
 Heck, L. P., 510
 Heckerman, D., 64, 72
 Hedayatnia, B., 495
 Hedberg, N., 421
 Heikkilä, J., 167, 296
 Heim, I., 308, 323, 440
 Heinz, J. M., 543
 Hellmann, S., 334, 476
 Hellrich, J., 400
 Hemphill, C. T., 203
 Henderson, D., 142
 Henderson, J., 354, 516
 Henderson, M., 503
 Henderson, P., 495, 507, 514
 Hendler, J. A., 321
 Hendrickson, C., 166
 Hendrickx, I., 352
 Hendrix, G. G., 391
 Henrich, V., 362
 Hepple, M., 347
 Herdan, G., 12, 28
 Hermann, K. M., 483, 485
 Hermjakob, U., 298, 468
 Hernault, H., 462
 Hicks, T., 462
 Hidey, C., 460
 Higgins, H., 543
 Hilf, F. D., 495, 515

- Hill, F., 96, 118, 438
 Hindle, D., 268, 270, 271
 Hinkelmann, E. A., 516
 Hinrichs, E., 362
 Hinton, G. E., 115, 134,
 137, 142, 175, 190
 Hirschberg, J., 13, 506, 509,
 511, 516, 527–529
 Hirschman, L., 73, 352,
 435, 436, 441, 472,
 485, 512
 Hirst, G., 96, 370, 387, 442,
 445, 457, 460, 462,
 463
 Hitzeman, J., 463
 Hjelmslev, L., 120
 Hlaváčová, J., 294
 Hobbs, J. R., 350–352, 440,
 443, 462
 Hochreiter, S., 184, 190
 Hockenmaier, J., 228
 Hoff, M. E., 142
 Hoffmann, P., 64, 397
 Hofmann, T., 121
 Hollenback, J., 542
 Holliman, E., 11, 495
 Holtzman, A., 393, 413
 Hon, H.-W., 544
 Hopcroft, J. E., 208
 Hopely, P., 166, 485
 Horvitz, E., 64, 72
 Hou, Y., 421, 425
 House, A. S., 543
 Householder, F. W., 167
 Hovanyecz, T., 513
 Hovy, D., 73, 360
 Hovy, E., 368
 Hovy, E. H., 99, 190, 277,
 352, 371, 403, 425,
 435, 436, 442, 457,
 460, 462, 468
 Howard, R. E., 142
 Hu, M., 64, 397, 403
 Hu, Z., 462
 Huang, E. H., 119
 Huang, L., 286
 Huang, X., 17, 544
 Huang, Z., 352
 Hubbard, W., 142
 Huddleston, R., 209, 230
 Hudson, R. A., 295
 Huffman, S., 352
 Hughes, M., 19
 Hull, D. A., 93, 121
 Hulteen, E. A., 513
 Hume, E., 542
 Hunter, J., 462
 Hunter, P., 505
 Hutto, C. J., 514
 Huval, B., 352
 Hwang, A., 460
 Hwang, J. D., 360
 Hymes, D., 391
 Iacobacci, I., 365
 Iida, M., 441
 Iida, R., 441
 Ingria, R., 268, 270, 343,
 345, 347, 515
 Inui, K., 441
 Irons, E. T., 244
 Irsoy, O., 190
 Irving, G., 137
 Isard, M., 137
 Isbell, C. L., 496
 Ishizuka, M., 462
 ISO8601, 344, 345
 Israel, D., 350–352
 Issar, S., 501
 Iyyer, M., 122, 363, 483
 Izquierdo, R., 362
 Jackel, L. D., 142
 Jackendoff, R., 230, 315
 Jacobs, P. S., 352
 Jacobson, N., 166
 Jaech, A., 73
 Jafarpour, S., 496
 Jang, S. B., 344
 Jat, S., 366
 Jauhar, S. K., 368
 Jauhainen, T., 73
 Jauvin, C., 111, 121, 138
 Javitt, D. C., 445
 Jefferson, G., 489, 490
 Jefferreys, H., 53
 Jelinek, F., 45, 53, 157, 250,
 268, 270, 271, 563
 Ji, H., 352, 417, 436
 Ji, Y., 449, 462, 497
 Jia, R., 472, 473
 Jia, Y., 137
 Jinová, P., 277, 296
 Johannsen, A., 73
 Johansson, R., 296, 392
 Johansson, S., 230
 Johnson, C. R., 379, 380
 Johnson, K., 542, 543
 Johnson, M., 19, 257, 315,
 360
 Johnson, W. E., 54
 Johnson-Laird, P. N., 379
 Johnston, M., 511
 Jones, M. P., 121
 Jones, R., 338, 352, 506
 Jones, S. J., 513
 Jones, T., 13
 Joos, M., 94, 98, 120
 Jordan, M., 190
 Jordan, M. I., 80, 121
 Jorge, A., 364
 Joshi, A., 447
 Joshi, A. K., 166, 223, 230,
 246, 271, 272, 441,
 444, 447, 452, 453,
 463, 485
 Joshi, M., 433, 483
 Joty, S., 462, 463
 Joulin, A., 121
 Jozefowicz, R., 137, 498
 Juang, B. H., 563
 Jun, S.-A., 529
 Jurafsky, D., 13, 34, 40, 65,
 99, 106, 117, 118,
 121, 165, 339, 340
 Khudanpur, S., 121, 176
 Kibble, R., 442
 Kiela, D., 121
 Kiesling, S., 542
 Kilgarriff, A., 365
 Kim, D., 510
 Kim, G., 217, 219
 Kim, S. M., 403
 Kim, S. N., 352
 King, L. A., 409
 Kingma, D., 137
 Kingsbury, P., 190, 392
 Kintsch, W., 121, 322, 452,
 457, 463
 Kiparsky, P., 373
 Kipper, K., 376, 392
 Kirchhoff, K., 506
 Kiritchenko, S., 399
 Kiss, T., 22
 Kit, C., 392
 Klapaitis, I. P., 369
 Klavans, J. L., 268, 270
 Kleene, S. C., 27
 Klein, D., 70, 71, 73, 163,
 164, 167, 257, 258,
 266, 271, 427, 430,
 431, 437, 442
 Klein, E., 16, 17, 28, 216
 Klein, S., 166, 167, 484
 Clementiev, A., 392
 Klingner, J., 19
 Kneser, R., 46, 47
 Knight, K., 298
 Knippen, R., 344
 Knott, A., 463
 Kobilarov, G., 334, 476
 Kocijan, V., 438
 Kociský, T., 485
 Kočiský, T., 483
 Koehn, P., 48, 49, 54, 298,
 527
 Koenig, W., 543
 Kolářová, V., 277, 296
 Kombrink, S., 121
 Kordon, V., 463
 Korhonen, A., 96, 118, 368
 Kormann, D., 496
 Kornai, A., 28
 Kosinski, M., 409–411
 Kozareva, Z., 352
 Kraemer, H. C., 495
 Kratzer, A., 308, 323
 Krieger, M., 28
 Kriese, L., 459
 Krizhevsky, A., 137
 Krötzsch, M., 334
 Krovetz, R., 22
 Kruskal, J. B., 28, 563
 Kübler, S., 296, 297
 Kučera, H., 11, 166
 Kudlur, M., 137
 Kudo, T., 19, 20, 296
 Kuhlmann, M., 297
 Kuhn, G. M., 190
 Kuhn, J., 430
 Kuhn, S., 463
 Kuksa, P., 111, 121, 190,
 352, 392

- Kulkarni, R. G., 506
 Kullback, S., 387
 Kumar, S., 366
 Kumlien, J., 352
 Kummerfeld, J. K., 427
 Kuno, S., 244
 Kuperberg, G. R., 445
 Kuperman, V., 398
 Kupiec, J., 166
 Kurian, G., 19
 Kurita, K., 439
 Kwak, H., 400
 Kwak, S., 294
 Kwan, J. L. P., 485
 Kwatra, S., 495
 Kwiatkowski, T., 485
 Laan, F., 362
 Labov, W., 93
 Lacy, L. Y., 543
 Ladd, D. R., 526
 Ladefoged, P., 543
 Ladusaw, W. A., 543
 Lafferty, J., 414
 Lafferty, J. D., 93, 163, 271,
 272, 352
 Laham, D., 121
 Lai, A., 445, 462
 Laird, N. M., 558
 Lakoff, G., 315, 322, 389,
 459
 Lal, T. N., 414
 Lally, A., 485
 Lamblin, P., 142
 Lample, G., 180, 331, 352
 Landauer, T. K., 118, 120,
 121, 457, 513
 Landes, S., 362
 Lando, T., 294
 Lang, J., 392
 Lapata, M., 388, 389, 392,
 444, 452, 454–457,
 462, 463
 Lapesa, G., 121
 Lappin, S., 426, 441
 Lari, K., 253
 Larochelle, H., 142
 Lascarides, A., 462, 463
 Laughery, K., 322, 476, 484
 Lauscher, A., 368
 Lazo, M., 347
 La Polla, R., 230
 LDC., 541
 Le, Q. V., 19, 497
 Leacock, C., 362
 Leass, H., 426, 441
 LeCun, Y., 142
 Lee, A., 447
 Lee, C.-H., 515
 Lee, D. D., 121
 Lee, H., 426, 427, 431, 442,
 462
 Lee, K., 19, 122, 363, 383,
 384, 392, 427,
 430–432, 434, 435,
 442, 475, 485
 Lee, L., 72, 73, 414, 460,
 462
- Leech, G., 167, 230
 Lehiste, I., 543
 Lehmann, J., 334, 476
 Lehnert, W. G., 352, 441
 Leibler, R. A., 387
 Lemon, O., 516
 Lengerich, B., 368
 Lenz, B., 368
 Leonard, B., 438
 Lerer, A., 137
 Lertpradit, S., 294
 Lesk, M. E., 370
 Leskovec, J., 117, 400–402
 Leung, H., 294, 512
 Leuski, A., 496, 498
 Levenberg, J., 137
 Levenshtein, V. I., 23
 Levesque, H., 437, 438
 Levesque, H. J., 322, 516
 Levin, B., 376, 391, 392
 Levin, E., 515, 516
 Levine, Y., 368
 Levinson, S. C., 515
 Levow, G.-A., 506, 508,
 513
 Levy, J. P., 121
 Levy, O., 110, 116, 121,
 122, 433, 438
 Lewis, C., 512
 Lewis, D. L., 73, 352
 Lewis, M., 266, 383, 384,
 392, 427, 430, 431,
 434, 435, 442
 Li, A., 542
 Li, D., 491, 492, 496
 Li, H., 496, 497
 Li, J., 17, 99, 294, 457, 458,
 462, 497, 498
 Li, M., 516
 Li, Q., 462
 Li, R., 457, 462
 Li, S., 449, 462
 Li, S.-W., 507
 Li, T., 462
 Li, X., 17, 468, 470
 Li, Y., 331, 392
 Li, Z., 118, 392, 496
 Liang, P., 472, 473, 478,
 479, 483
 Liberman, A. M., 543
 Liberman, M. Y., 268, 270,
 516
 Lieberman, H., 321
 Lieberman Aiden, E., 117
 Light, M., 472, 485
 Lim, D. C. Y., 428, 429,
 441
 Lin, D., 270, 296, 393, 427,
 441
 Lin, J., 468, 472
 Lin, Y., 117
 Lin, Z., 137, 445, 451, 456,
 463, 516
 Lindén, K., 73
 Lindsey, R., 322
 Ling, W., 190
 Ling, X., 437
 Liscome, J., 505
- Lison, P., 495
 Litkowski, K. C., 392
 Litman, D. J., 490, 506,
 509, 511, 512, 516
 Littman, J., 343–345
 Littman, M., 400, 402
 Liu, A., 511
 Liu, B., 64, 73, 397, 403
 Liu, C.-W., 497, 498
 Liu, T., 366, 392
 Liu, T.-H., 291
 Liu, Y., 452
 Liwicki, M., 190
 Llorens, H., 353
 Lochbaum, K. E., 120, 516
 Logeswaran, L., 462
 Long, T., 457, 458, 462
 Loper, E., 16, 17, 28
 Lopez-Gazpio, I., 119
 Lopyrev, K., 472, 483
 Louis, A., 462, 463
 Loureiro, D., 364
 Louviere, J. J., 399
 Lovins, J. B., 27
 Lowe, J. B., 379
 Lowe, R., 495, 514
 Lowe, R. T., 497, 498
 Lu, Z., 496, 497
 Luhn, H. P., 105
 Lui, M., 65, 73
 Lukasiewicz, T., 438
 Luo, F., 366
 Luo, X., 435, 436
 Luotoalahti, J., 294
 Luís, T., 190
 Lyons, J., 323
 Lyons, R. C., 544
 Lytel, D., 370
- Ma, X., 190, 352
 Maas, A., 368
 MacBride, J., 425
 MacCartney, B., 296
 Macherey, K., 19
 Macherey, W., 19
 MacIntyre, R., 217, 219
 Mackenzan, V., 294
 MacLeod, C., 378
 Macy, M., 514
 Madhu, S., 370
 Madotto, A., 516
 Magerman, D. M., 221,
 271, 278
 Maharjan, S., 13
 Maier, W., 297
 Maiorano, S., 468
 Mairesse, F., 395, 411, 503,
 504, 516
 Manandhar, S., 369
 Mandl, M., 294
 Mané, D., 137
 Mani, I., 343–345
 Mann, W. C., 445, 462
 Manning, C. D., 22, 72, 73,
 80, 111, 115, 116,
 119, 121, 122, 147,
 163–165, 167, 190,
 217, 253, 257, 258,
- 266, 271, 275, 286,
 294, 296, 345, 352,
 353, 392, 414, 430,
 442, 474, 478
 Manurung, R., 294
 Marcinkiewicz, M. A., 146,
 217, 219, 253, 277,
 296
 Marcu, D., 445–447, 449,
 451, 462, 463
 Marcus, M. P., 146, 179,
 190, 217, 219, 253,
 268, 270, 271, 277,
 296, 371, 391, 425,
 436
 Marcus, S., 109
 Marheinecke, K., 294
 Marinov, S., 296
 Marixalar, M., 119
 Markert, K., 421, 425
 Markov, A. A., 53, 563
 Markovitch, S., 109
 Marley, A. A. J., 399
 Marlin, B. M., 352
 Maron, M. E., 72
 Márquez, L., 296, 384, 392,
 393
 Marshall, C., 516
 Marshall, I., 166
 Marsi, E., 296
 Martí, M. A., 296, 420,
 425, 442
 Martin, J. H., 121, 190,
 371, 392
 Martin, R., 516
 Martinez, D., 393
 Martschat, S., 427
 Marujo, L., 190
 Marx, M., 508, 513
 Masterman, M., 322, 370
 Mathis, D. A., 190
 Matias, Y., 118
 Matsumoto, Y., 296, 441
 Mausam, 393
 Mausam., 352
 McCallum, A., 72, 93, 163,
 352, 437
 McCarthy, J., 229
 McCarthy, J. F., 441
 McCaw, J., 31
 McCawley, J. D., 230, 322,
 323
 McClelland, J. L., 142, 190
 McClosky, D., 22
 McConlogue, K., 484
 McConnell-Ginet, S., 323
 McCord, M. C., 485
 McCoy, K. F., 31
 McCulloch, W. S., 123, 141
 McDaniel, J., 11, 495
 McDonald, R., 147, 165,
 217, 275, 290, 296,
 297, 414
 McEnergy, A., 167
 McFarland, D. A., 409
 McGhee, D. E., 117, 118
 McGuiness, D. L., 321

- McKeown, K., 401, 402,
 414, 452, 460
- Meek, C., 473, 483
- Mehl, M. R., 409
- Melamud, O., 363
- Mel'uk, I. A., 295
- Melis, G., 483
- Mellish, C., 463
- Mendonça, G., 294
- Meng, Y., 17
- Mercer, R. L., 45, 53, 157,
 166, 271
- Merialdo, B., 166
- Mesgar, M., 463
- Meteer, M., 166, 516
- Metsis, V., 72
- Meyers, A., 296, 378, 392
- Micciulla, L., 425
- Michael, J., 438
- Michaelis, L. A., 254
- Michel, J.-B., 117
- Mihalcea, R., 119, 367,
 371, 418, 436, 437
- Mikheev, A., 329
- Mikolov, T., 54, 111, 116,
 119, 121, 176, 190
- Mikulová, M., 277, 296
- Mildiú, R. L., 430
- Miller, C. A., 526
- Miller, G. A., 39, 53, 362,
 414, 543
- Miller, S., 352, 515
- Milne, D., 437
- Miltzakaki, E., 447
- Minsky, M., 72, 126, 142,
 391
- Mintz, M., 339, 352
- Mirovský, J., 277, 296
- Missilä, A., 294
- Mitchell, M., 497
- Mitchell, T. M., 122
- Mitkov, R., 442
- Miwa, M., 336, 352
- Mochales Palau, R., 459
- Moens, M., 329, 461
- Moens, M.-F., 392, 393,
 459
- Mohamed, A., 190
- Mohammad, S. M.,
 397–399
- Moldovan, D., 371
- Monga, R., 137
- Monroe, B. L., 406–408
- Monroe, W., 497, 498
- Montague, R., 322
- Monz, C., 469
- Mooney, R. J., 478
- Moore, S., 137
- Moosavi, N. S., 435
- Morante, R., 392
- Morey, M., 449, 451, 452
- Morgan, A. A., 352
- Morgan, N., 34, 142, 544
- Morgenstern, L., 438
- Morimoto, T., 516
- Morin, F., 54, 121
- Morris, J., 457
- Morris, M. R., 31
- Morris, W., 355
- Moschitti, A., 384, 425,
 426, 436
- Mosteller, F., 58, 72
- Moszkowicz, J., 347
- Mota, N. B., 445
- Mozer, M. C., 190
- Mrkšić, N., 368, 505, 510
- Mulcaire, G., 73
- Muller, P., 449, 451, 452
- Murdock, J. W., 485
- Muresan, S., 459, 460
- Murphy, B., 122
- Murphy, K. P., 73, 93
- Murray, D., 137
- Musi, E., 459, 460
- Myers, G., 461
- Nádas, A., 54
- Nagao, M., 272
- Nagata, M., 516
- Nagy, P., 514
- Nakov, P., 352
- Naradowsky, J., 438
- Narayanan, A., 118, 439
- Narayanan, S. S., 508
- Nash-Webber, B. L., 391,
 440, 485
- Naur, P., 229
- Navigli, R., 361, 362, 365,
 367, 369, 371
- Nayak, N., 510
- Nedoluzhko, A., 277, 294,
 296
- Needleman, S. B., 563
- Neff, G., 514
- Nenkova, A., 436, 452, 462,
 463
- Neumann, M., 122, 363
- Newman, E. B., 533
- Ney, H., 46, 47, 73, 251
- Ng, A. Y., 80, 119, 121,
 340, 352, 371
- Ng, H. T., 365, 384, 428,
 429, 441, 448, 451,
 456, 463, 485
- Ng, R. T., 462
- Ng, V., 426–431, 436, 438,
 441, 442
- Nguyen, D. T., 463
- Nguyen, K. A., 368
- Nicely, P. E., 543
- Nichols, E., 190
- Niculae, V., 460
- Nie, A., 451, 463
- Nielsen, J., 513
- Nielsen, M. A., 142
- Nigam, K., 72, 93, 352
- Nilsson, J., 296
- Nilsson, N. J., 516
- Nissim, M., 421
- NIST, 28, 541
- Nitisaroj, R., 294
- Nitta, Y., 109
- Nivre, J., 147, 165, 217,
 275, 280, 286, 288,
 290, 294, 296, 297,
 392
- Niwa, Y., 109
- Noreen, E. W., 70
- Norman, D. A., 322, 391,
 489, 498, 499, 515
- Norouzi, M., 19
- Norvig, P., 128, 244, 304,
 322
- Nosek, B. A., 118
- Noseworthy, M., 498
- Novick, D. G., 513
- Nunes, J. H. T., 516
- O'Connor, B., 13, 28, 40,
 65
- O'Hara, T. P., 394
- O'Shaughnessy, D., 544
- Oberlander, J., 463
- Och, F. J., 49
- Oettinger, A. G., 244
- Oflazer, K., 164
- Ogilvie, D., 64, 397, 414
- Ojala, S., 294
- Okurowski, M. E., 447, 462
- Olah, C., 137
- Oravec, C., 164
- Ordonez, V., 118, 438, 439
- Ortiz, C. L., 438
- Orwant, J., 117
- Osborne, M., 49, 167
- Osgood, C. E., 97–99, 120,
 397, 414
- Osindero, S., 142
- Ostendorf, M., 73, 506,
 527, 528
- Ozertem, U., 506
- Ó Séaghdha, D., 352, 505,
 510
- O'Connor, B., 459
- Packard, D. W., 27
- Padnos, D., 368
- Padó, S., 296, 352
- Palioras, G., 72
- Paliwal, K. K., 182, 190
- Palmer, D., 28
- Palmer, M., 190, 277, 278,
 288, 296, 298, 362,
 371, 376, 392, 425,
 436
- Palmucci, J., 166
- Pan, Y., 485
- Paneeva, J., 295
- Panevová, J., 277, 296
- Pang, B., 72, 73, 414
- Pao, C., 512
- Paolino, J., 514
- Papert, S., 126, 142
- Pappas, N., 354
- Paradiso, A., 31
- Pareek, A., 439
- Parikh, A., 485
- Paritosh, P., 334, 476
- Park, J., 460
- Parsons, T., 312, 322
- Partee, B. H., 322
- Pasca, M., 468, 469, 471
- Paszke, A., 137
- Patil, N., 19
- Patwardhan, S., 485
- Pearl, C., 513
- Pedersen, J., 73, 93, 116,
 121
- Pedersen, T., 371
- Peirson, Y., 426, 442
- Peldszus, A., 459, 460
- Peng, N., 336, 352
- Peng, S., 452
- Penn, G., 373
- Pennacchiotti, M., 352
- Pennebaker, J. W., 64, 398,
 399, 409
- Pennington, C., 31
- Pennington, J., 111, 115,
 116, 121, 190, 474
- Percival, W. K., 229
- Pereira, F. C. N., 163, 296,
 352
- Perkowitz, M., 166
- Perlis, A. J., 229
- Perrault, C. R., 516
- Peters, M., 122, 363
- Peters, S., 322
- Peterson, G. E., 543
- Petrie, T., 563
- Petrov, S., 117, 147, 165,
 217, 258, 271, 275,
 294, 296, 297
- Petruck, M. R. L., 379, 380
- Phillips, A. V., 484
- Phillips, J., 344
- Phillips, M., 512
- Picard, R. W., 394
- Piepenbrock, R., 539
- Pieraccini, R., 505, 515,
 516
- Pierrehumbert, J. B., 528,
 529
- Pilehvar, M. T., 119, 365,
 367, 371
- Pineau, J., 495, 497, 498,
 514, 516
- Pitler, E., 294, 452, 463
- Pitrelli, J. F., 528
- Pitt, M. A., 542
- Pitts, W., 123, 141
- Plank, B., 73
- Plato, 543
- Plaut, D. C., 115
- Plutchik, R., 396, 397
- Poesio, M., 419, 420, 425,
 441, 442, 463
- Poláková, L., 277, 296
- Polanyi, L., 462
- Polifroni, J., 512
- Pollard, C., 220, 221, 223,
 229, 246, 441, 453
- Ponti, E. M., 368
- Ponzerotto, S. P., 367, 437
- Poon, H., 336, 352
- Popat, A. C., 49
- Popel, M., 294
- Popescu, A.-M., 352
- Popescu-Belis, A., 354
- Popovici, D., 142
- Porter, M. F., 21, 22
- Pothast, M., 294

- Potts, C., 368, 404–406,
441
- Pouzyrevsky, I., 48, 49, 54
- Pow, N., 497
- Pradhan, S., 190, 369, 384,
392, 425, 426, 435,
436, 448
- Prager, J. M., 485
- Prakash, S., 371
- Prange, J., 360
- Prasad, R., 447
- Prasettio, M. C., 393, 413
- Prendinger, H., 462
- Prescher, D., 393
- Price, P. J., 506, 527, 528
- Prince, E., 420, 421, 463
- Propp, V., 459
- Przepiórkowski, A., 297
- Pu, X., 354
- Pullum, G. K., 209, 216,
230, 543
- Purver, M., 511
- Pustejovsky, J., 343–345,
347, 353, 371, 442
- Pyysalo, S., 147, 165, 217,
275, 294, 296
- Qi, P., 294
- Qin, B., 392
- Qin, L., 462
- Qiu, X., 17
- Qiu, Z. M., 485
- Quillian, M. R., 322, 370
- Quinn, K. M., 406–408
- Quirk, C., 336, 352
- Quirk, R., 230
- Rabiner, L. R., 544, 550,
560, 561, 563
- Radev, D., 347, 442, 462
- Radford, A., 230
- Raganato, A., 365, 371
- Raghavan, P., 73, 122
- Raghunathan, K., 442
- Rahman, A., 427, 429, 430,
438, 441
- Rajpurkar, P., 472, 473, 483
- Ramshaw, L., 425, 436
- Ramshaw, L. A., 166, 179,
190, 270, 277, 371
- Ranganath, R., 409
- Rangarajan, S., 442
- Raphael, B., 322
- Rappaport Hovav, M., 376
- Rashkin, H., 392, 393, 412,
413, 516
- Ratinov, L., 121, 437
- Ratnaparkhi, A., 93, 167,
271, 272
- Rau, L. F., 352
- Raux, A., 503
- Ravichandran, D., 468
- Raymond, W. D., 542
- Recasens, M., 420, 425,
435, 436, 438, 439,
441, 442
- Reddy, S., 294
- Reed, C., 459
- Reeves, R., 378
- Rehder, B., 121
- Rehm, G., 294
- Reichert, R., 73, 96, 118
- Reichenbach, H., 314
- Reichert, T. A., 563
- Reichman, R., 421
- Reiss, F., 331
- Reiss, F. R., 331
- Renshaw, E., 484
- Resnik, P., 272, 371, 387,
388, 393
- Reynar, J. C., 272
- Ribarov, K., 296
- Ribeiro, S., 445
- Richardson, J., 20
- Richardson, M., 483, 484
- Riedel, S., 296, 352
- Rieger, C., 370
- Ries, K., 516
- Riesa, J., 19
- Riesbeck, C. K., 370
- Riezler, S., 393
- Rigau, G., 119
- Riloff, E., 338, 352, 392,
397, 414, 415, 419,
427, 441, 485
- Ritter, A., 352, 393,
495–498
- Rivlin, E., 118
- Robaldo, L., 447
- Robins, R. H., 543
- Rocci, A., 459
- Rodriguez, P., 485
- Rohde, D. L. T., 115
- Rohde, H., 441
- Rojas-Barahona, L. M., 368
- Romanov, L., 352
- Rooth, M., 271, 393
- Rosenblatt, F., 142
- Rosenfeld, R., 93
- Rosenthal, S., 460
- Rosenzweig, J., 365
- Roth, D., 427, 430, 431,
436, 437, 468, 470
- Roth, R., 297
- Rothe, S., 415
- Roukos, S., 268, 270–272
- Routledge, B. R., 407, 408
- Rowe, G., 459
- Roy, N., 516
- Rozovskaya, A., 427
- Rubin, D. B., 558
- Rubin, G. M., 166
- Rudinger, R., 438
- Rudnicky, A. I., 509
- Ruhi, U., 542
- Rumelhart, D. E., 134, 142,
175, 190, 322
- Rumshisky, A., 344
- Ruppenhofer, J., 379, 380,
392
- Ruppini, E., 118
- Rush, A. M., 427, 430, 431,
442
- Russell, J. A., 396
- Russell, S., 128, 304, 322
- Rutherford, A., 448, 463
- Rutishauser, H., 229
- Sabharwal, A., 484
- Sacks, H., 489
- Sag, I. A., 216, 220, 221,
223, 229, 230, 246,
511, 516
- Sagae, K., 286, 462
- Saghir, H., 457, 458, 462
- Sahami, M., 64, 72
- Sakoe, H., 563
- Salakhutdinov, R., 363
- Salakhutdinov, R. R., 137
- Salant, S., 485
- Saligrama, V., 117, 118,
439
- Salomaa, A., 271
- Salton, G., 100, 120
- Samdani, R., 427, 430
- Samelson, K., 229
- Sammons, M., 427
- Sampson, G., 167
- Samuelsson, C., 159, 167
- Sanfilippo, A., 270
- Sankoff, D., 93, 563
- Santorini, B., 146, 217,
253, 268, 270, 277,
296
- Sap, M., 393, 413
- Sauri, R., 343, 345, 347
- Sauri, R., 344
- Saxena, K., 366
- Scha, R., 462
- Schabes, Y., 271, 272
- Schaffer, R. W., 544
- Schalkwyk, J., 513
- Schank, R. C., 322, 349,
379, 391, 392
- Schapire, R. E., 166, 272
- Schasberger, B., 217, 219
- Schatzmann, J., 503, 504
- Scheffczyk, J., 379, 380
- Schegloff, E. A., 489
- Scherer, K. R., 394, 395,
409, 410
- Schiebinger, L., 118, 416,
439
- Schilder, F., 347
- Schmandt, C., 513
- Schmelzenbach, M., 392
- Schmidhuber, J., 184, 190
- Schmolze, J. G., 322
- Schneider, J., 461
- Schneider, N., 298, 360
- Schoenick, C., 484
- Schölkopf, B., 414
- Scholz, M., 296
- Schone, P., 121
- Schönfinkel, M., 308
- Schreiner, M. E., 121
- Schulte im Walde, S., 368
- Schulz, H., 438
- Schuster, M., 19, 137, 182,
190
- Schuster, S., 294
- Schütze, H., 73, 93, 116,
121, 122, 166, 253,
368, 371, 388, 415
- Schwartz, H. A., 409–411
- Schwartz, J. L. K., 117, 118
- Schwartz, R., 166, 352, 515
- Schwarz, J., 483
- Schwenk, H., 54, 121, 185
- Séaghda, D. O., 393
- Séaghda, D. Ó., 368
- Sebastiani, F., 403
- Seddah, D., 297
- See, A., 347
- Segal, J., 34
- Sekine, S., 270
- Selfridge, J. A., 39
- Semeraro, G., 366
- Senécal, J.-S., 54, 121
- Seneff, S., 512, 541
- Sennrich, R., 18, 20
- Seo, M., 485
- Serban, I. V., 495, 497, 498
- Sethi, A., 506, 516
- Sethi, R., 244
- Setzer, A., 343, 345, 347
- Seung, H. S., 121
- Søgaard, A., 73, 167, 190,
447, 462
- Sgall, P., 295
- Shah, A., 19, 409–411
- Shaked, T., 352
- Shalev-Shwartz, S., 368
- Shang, L., 497
- Shannon, C. E., 39, 53, 177
- Sharir, O., 368
- Sharma, D., 288
- Shashua, A., 368
- Shattuck-Hufnagel, S., 527
- Shaw, G. B., 543
- Sheil, B. A., 244
- Sheinwald, D., 485
- Shepherd, J., 414, 415
- Shi, T., 497, 498
- Shi, Z., 17
- Shieber, S. M., 427, 430,
431, 442
- Shima, H., 485
- Shimada, A., 294
- Shin, J., 516
- Shlens, J., 137
- Shockley, L., 543
- Shoham, Y., 368
- Shoup, J. E., 519
- Shriberg, E., 506, 516
- Shrivastava, M., 288
- Shum, H.-Y., 491, 492, 496
- Siddharthan, A., 461
- Sidner, C. L., 463, 516
- Sigman, M., 445
- Silveira, N., 147, 165, 217,
275, 296
- Silverman, K., 528
- Simi, M., 294
- Simma, A., 441
- Simmons, R. F., 166, 167,
322, 370, 381, 391,
484, 485
- Simons, G. F., 13
- Singer, Y., 163, 164, 166,
167, 272
- Singh, A., 438

- Singh, S., 392, 412, 413, 437, 496
 Singh, S. P., 516
 Sinha, K., 514
 Sleator, D., 272, 296
 Slezak, D. F., 445
 Slocum, J., 391
 Small, S. L., 370
 Smith, E. M., 516
 Smith, J., 19
 Smith, M., 64, 397, 414
 Smith, N. A., 73, 368, 407, 408, 459
 Smith, V. L., 517
 Smolensky, P., 142
 Snow, R., 339, 340, 352, 371
 Snyder, B., 362, 456
 Socher, R., 111, 115, 116, 119, 121, 190, 352, 474
 Soderland, S., 340, 341, 352, 478
 Solan, Z., 118
 Solorio, T., 13
 Somasundaran, S., 445
 Song, Z., 542
 Soon, W. M., 428, 429, 441
 Sordoni, A., 497
 Soricut, R., 462
 Sparck Jones, K., 106, 120, 370, 371
 Spencer, T., 292
 Spitskovsky, V. I., 437, 482
 Sporleder, C., 392, 462, 463
 Srikanth, V., 360
 Srinivas, B., 272
 Srivastava, N., 137
 Stab, C., 459, 460
 Stalnaker, R. C., 489
 Stamatasos, E., 73
 Stede, M., 459–461, 463
 Steedman, M., 167, 223, 228, 266, 421, 529
 Steiner, B., 137
 Stent, A., 297
 Štěpánek, J., 296
 Stern, A., 360
 Stetina, J., 272
 Stevens, K., 19
 Stevens, K. N., 543, 544
 Stevens, S. S., 533
 Stevenson, R., 463
 Stevenson, S., 392
 Stifelman, L. J., 513
 Stillwell, D., 409–411
 Stoicks, 143
 Stolcke, A., 34, 49, 54, 271, 516
 Stoltz, W. S., 166
 Stone, P., 64, 397, 414, 496
 Stone, P. J., 370
 Stoyanchev, S., 511
 Straka, M., 294
 Stranák, P., 296
 Streeter, L., 120
 Strnadová, J., 294
 Strötgen, J., 345, 353
 Strube, M., 421, 425, 427, 435–437, 441, 463
 Strunk, J., 22
 Strzalkowski, T., 268, 270
 Stuckardt, R., 420, 442
 Sturge, T., 334, 476
 Sturt, P., 442
 Stuttle, M., 516
 Su, J., 352, 441
 Su, P.-H., 368, 510
 Subba, R., 462
 Subramanian, S., 180, 331, 352, 436
 Suci, G. J., 97–99, 120, 397, 414
 Suendermann, D., 505
 Suh, J., 483
 Sui, Z., 366
 Suleiman, K., 438
 Sulubacak, U., 294
 Sun, X., 17
 Sundheim, B., 343, 345, 347, 350, 352
 Surdeanu, M., 22, 296, 352, 392, 393, 426, 427, 431, 442, 462
 Sutskever, I., 111, 121, 137
 Sutton, S., 513
 Svartvik, J., 230
 Sweet, H., 543
 Swerts, M., 506, 509
 Swier, R., 392
 Switzer, P., 120
 Székely, R., 378
 Szpakowicz, S., 352
 Tafjord, O., 484
 Tajchman, G., 34
 Taji, D., 294
 Takamura, H., 441
 Talbot, D., 49
 Talmor, A., 483
 Talukdar, P., 366
 Talukdar, P. P., 122
 Talwar, K., 137
 Tan, C., 460
 Tan, C. L., 441
 Tannen, D., 391
 Tannenbaum, P. H., 97–99, 120, 166, 397, 414
 Tarjan, R. E., 292
 Taylor, J., 334, 476
 Taylor, L., 441
 Taylor, P., 516
 Teh, Y.-W., 142
 Temperley, D., 272, 296
 Tengi, R. I., 362
 Teo, L. H., 485
 ter Meulen, A., 323
 Tesnière, L., 295, 391
 Tetreault, J., 297, 445, 462
 Tetreault, J. R., 441
 Teufel, S., 461
 The PDP Research Group, 190
 Thede, S. M., 167
 Thelen, M., 485
 Thibaux, R., 258, 271
 Thione, G. L., 462
 Thomas, J. A., 50, 51
 Thompson, C. W., 391
 Thompson, H., 391, 498, 499, 515
 Thompson, K., 27
 Thompson, R. A., 247
 Thompson, S. A., 445, 462
 Thomson, B., 368, 503–505, 510
 Thrax, D., 143
 Thrun, S., 516
 Tibshirani, R. J., 70, 73, 88, 93
 Ticrea, M., 442
 Tiedemann, J., 495
 Tillmann, C., 270
 Titov, I., 392
 Tomkins, S. S., 396
 Toutanova, K., 19, 122, 163, 164, 167, 336, 352, 475
 Trancoso, I., 190
 Traum, D., 496, 498
 Trichelaar, P., 438
 Trischler, A., 438
 Trnka, K., 31
 Tsarfaty, R., 147, 165, 217, 275, 296, 297
 Tseng, B.-H., 510
 Tseng, H., 165
 Tsvetkov, Y., 13, 40, 65, 360, 411, 439
 Tucker, P., 137
 Tur, G., 516
 Tür, G., 164
 Turian, J., 121
 Turner, M., 442
 Turney, P. D., 397, 399, 400, 402, 414
 Tyers, F. M., 294
 Tyson, M., 350–352
 Ullman, J. D., 208, 244, 278
 Ultes, S., 510
 Ungar, L. H., 409–411
 Uresová, Z., 294
 Uria, L., 119
 Uryupina, O., 425, 426, 436
 Uszkoreit, H., 294
 UzZaman, N., 353
 Vaithyanathan, S., 72, 414
 Valenzuela-Escarcega, M. A., 462
 van Benthem, J., 323
 van Deemter, K., 442
 Van Den Bosch, A., 362
 van der Maaten, L., 115
 Van Ess-Dykema, C., 516
 van Harmelen, F., 321
 van Merriënboer, B., 185
 van Rijsbergen, C. J., 67, 243, 270
 Van Valin, Jr., R. D., 230
 van Wijngaarden, A., 229
 Vandyke, D., 368, 510
 Vanhoucke, V., 137
 Vannella, D., 369
 van den Berg, M., 462
 Van Dijk, T. A., 452, 463
 Van Durme, B., 438
 Van Gompel, M., 362
 Van Ooyen, B., 41
 Vasudevan, V., 137
 Vauquois, B., 229
 Veilleux, N., 527
 Velikovich, L., 414
 Venditti, J. J., 529
 Vendler, Z., 315
 Venkataramani, V., 542
 Venkatesh, A., 495
 Verhagen, M., 344, 347, 353
 Vermeulen, P. J. E., 513
 Versley, Y., 297, 420, 442
 Viégas, F., 137
 Vieira, R., 419, 441
 Vilain, M., 435, 436, 441
 Villemonte de la Clérgerie, E., 297
 Vilnis, L., 437, 498
 Vincent, P., 94, 111, 121, 138
 Vincze, V., 297
 Vintsyuk, T. K., 563
 Vinyals, O., 19, 137, 497, 498
 Viterbi, A. J., 563
 Vodolazova, T., 362
 Volkmann, J., 533
 Volkova, S., 392, 412, 413
 Vossen, P., 362
 Voutilainen, A., 167, 296
 Vrandečić, D., 334
 Vu, N. T., 368
 Vulić, I., 368
 Vyas, N., 439
 Wade, E., 506
 Wagner, R. A., 24, 563
 Wagstaff, K., 441
 Wahlster, W., 321
 Walker, M. A., 395, 411, 441, 463, 490, 506, 508, 510–512, 516
 Wall, R. E., 322
 Wallace, D. L., 58, 72
 Wang, A., 438
 Wang, C., 448
 Wang, H., 448, 496
 Wang, S., 72, 80, 414
 Wang, T., 118, 438, 439
 Wang, W., 19, 118
 Wang, Y., 449, 462
 Ward, W., 392, 501
 Warden, P., 137
 Warmsley, D., 514
 Warriner, A. B., 398
 Wasow, T., 230
 Wattberg, M., 137
 Weaver, W., 370
 Webber, B. L., 417, 418, 421, 440, 447, 448, 463

- Weber, I., 514
 Weber, S., 495, 515
 Webster, K., 438, 439
 Weenink, D., 534, 542, 544
 Wegstein, J. H., 229
 Wehbe, L., 122
 Weinberger, D. R., 445
 Weinschenk, S., 508
 Weinstein, S., 441, 444,
 452, 453, 463
 Weischedel, R., 166, 277,
 352, 371, 425, 436
 Weischedel, R. M., 190
 Weizenbaum, J., 2, 10,
 491–493, 515
 Weld, D. S., 352, 433, 437,
 442, 483
 Wells, J. C., 526, 543
 Welty, C., 485
 Wen, T.-H., 368, 505, 510
 Werbos, P., 175
 Werbos, P. J., 175
 Wessels, L. F. A., 513
 Weston, J., 111, 121, 190,
 352, 392, 414, 427,
 431, 468, 473, 474,
 485
 Whiteside, J. A., 513
 Whitney, D., 507
 Whittaker, S., 490
 Wicke, M., 137
 Widrow, B., 142
 Wiebe, J., 64, 119, 394,
 397, 414
 Wierzbicka, A., 120
 Wightman, C. W., 528
 Wilensky, R., 516
 Wilkes-Gibbs, D., 516
 Wilks, Y., 322, 370, 387,
 391
 Williams, J., 392
 Williams, J. D., 503, 516
 Williams, R., 352
 Williams, R. J., 134, 142,
 175, 190
 Wilson, G., 343, 345
 Wilson, T., 64, 397
 Winograd, T., 322, 391,
 437, 440, 498, 499,
 515
 Winston, P. H., 391
 Wiseman, S., 427, 430,
 431, 442
 Witten, I. H., 54, 73, 93,
 437
 Wittgenstein, L., 98, 489,
 515
 Wixon, D. R., 513
 Wolf, A. K., 322, 476, 484
 Wolf, F., 462
 Wolfe, M. B. W., 121
 Woliński, M., 297
 Wong, A. K. C., 563
 Woodger, M., 229
 Woods, W. A., 322, 440,
 485
 Woodsend, K., 392
 Wootters, C., 34
 Wróblewska, A., 297
 Wu, F., 352
 Wu, Y., 19
 Wundt, W., 205, 229
 Wunsch, C. D., 563
 Xia, F., 278, 296
 Xia, Q., 366
 Xiang, B., 336, 352, 442
 Xing, E., 462
 Xu, P., 49, 457, 458, 462,
 516
 Xu, W., 190, 352, 392
 Xue, N., 277, 296, 384,
 392, 425, 426, 436,
 447, 448, 463
 Yallop, C., 543
 Yamada, H., 296
 Yan, Z., 496
 Yang, D., 460
 Yang, E., 137
 Yang, J., 449, 462
 Yang, X., 441
 Yang, Y., 73, 473, 483
 Yang, Z., 460
 Yankelovich, N., 508, 513
 Yao, L., 352
 Yarowsky, D., 363, 371,
 388
 Yarrington, D., 31
 Yasser, T., 28
 Yasunaga, M., 442
 Yates, A., 352
 Yatskar, M., 118, 438, 439,
 483
 Yeh, A. S., 352
 Yi, L., 542
 Yih, W.-t., 116, 119, 336,
 352, 473, 483
 Yngve, V. H., 244
 Yordanov, Y., 438
 Young, B., 378
 Young, C., 19
 Young, S., 505, 510
 Young, S. J., 253, 503, 504,
 510, 516
 Younger, D. H., 232, 244
 Yu, K., 352, 503, 504
 Yu, M., 368
 Yu, N., 449, 450, 462
 Yu, Y., 137, 452
 Yu, Z., 294
 Yuan, A., 17
 Yuret, D., 296
 Zacharski, R., 421
 Zampieri, M., 73
 Zapirain, B., 393
 Zavrel, J., 272
 Zeldes, A., 452
 Zelle, J. M., 478
 Zeman, D., 147, 165, 217,
 275, 294, 296
 Zettlemoyer, L., 122, 352,
 363, 383, 384, 392,
 427, 430–435, 437,
 442, 478, 479, 483
 Zhai, C., 73
 Zhang, J., 352, 472, 483
 Zhang, L., 73, 485
 Zhang, M., 352, 449, 450,
 462
 Zhang, R., 442
 Zhang, Y., 286, 296, 384,
 425, 426, 436
 Zhang, Z., 462
 Zhao, H., 392, 462
 Zhao, J., 118, 438, 439
 Zheng, F., 542
 Zheng, J., 437
 Zheng, X., 137
 Zhong, Z., 365, 384
 Zhou, B., 336, 352
 Zhou, D., 414
 Zhou, G., 352, 392, 441
 Zhou, J., 190, 392, 496
 Zhou, L., 442, 491, 492,
 496
 Zhou, M., 496
 Zhou, Y., 118, 447
 Zhu, H., 331
 Zhu, X., 414
 Zhu, Y., 452
 Zielinska, V., 378
 Zikánová, Š., 277, 296
 Zilles, L., 437, 442
 Zou, J., 117, 118, 439
 Zue, V. W., 512, 541
 Zweig, G., 116, 119

Subject Index

- λ -reduction, 308
 *?, 7
 +?, 7
 .wav format, 531
 10-fold cross-validation, 69
 → (derives), 205
 ^, 58
 * (RE Kleene *), 5
 + (RE Kleene +), 5
 . (RE any character), 5
 \$ (RE end-of-line), 5
 (RE precedence symbol), 6
 [(RE character disjunction), 4
 \B (RE non word-boundary), 6
 \b (RE word-boundary), 6
] (RE character disjunction), 4
 ^ (RE start-of-line), 5
 [^] (single-char negation), 4
 ∃ (there exists), 306
 ∀ (for all), 306
 ==> (implies), 309
 λ -expressions, 308
 λ -reduction, 308
 ∧ (and), 306
 ¬ (not), 306
 ∨ (or), 309
 4-gram, 35
 4-tuple, 207
 5-gram, 35
 A-D conversion, 530
 AAC, 31
 AAVE, 13
 abduction, 311
 ABox, 316
 ABSITY, 370
 absolute discounting, 46
 absolute temporal expression, 343
 abstract word, 398
 accent
 nuclear, 525
 accented syllables, 525
 accessible, 421
 accessing a referent, 416
 accomplishment
 expressions, 315
 achievement expressions, 315, 315
 acknowledgment speech act, 489
 activation, 124
 activity expressions, 315, 315
 add gate, 185
 add-k, 44
 add-one smoothing, 42
 adjacency pairs, 489
 adjective, 144, 212
 adjective phrase, 212
 adjunction in TAG, 230
 adverb, 144
 days of the week coded as noun instead of, 145
 degree, 145
 directional, 145
 locative, 145
 manner, 145
 syntactic position of, 212
 temporal, 145
 adversarial evaluation, 498
 affective, 394
 affix, 21
 affricate sound, 523
 agent, as thematic role, 374
 agglomerative clustering, 369
 AIFF file, 531
 ALGOL, 229
 algorithm
 CKY, 234
 forward, 554
 forward-backward, 562
 inside-outside, 253
 Kneser-Ney discounting, 46
 Lev, 366
 minimum edit distance, 26
 n-gram tiling for question answering, 472
 naïve Bayes classifier, 58
 pointwise mutual information, 108
 probabilistic CKY, 251
 semantic role labeling, 381
 Simplified Lev, 366
 TextTiling, 457
 unsupervised word sense disambiguation, 368
 Viterbi, 153, 555
 alignment, 23
 minimum cost, 25
 of transcript, 541
 string, 23
 via minimum edit distance, 25
 all-words task in WSD, 362
 Allen relations, 347
 allophonic rules, 542
 alveolar sound, 522
 ambiguity
 amount of part-of-speech in Brown corpus, 148
 attachment, 233
 coordination, 233, 234, 256
 in meaning representations, 299
 of referring expressions, 418
 part-of-speech, 148
 PCFG in, 249
 prepositional phrase attachment, 255
 resolution of tag, 148
 word sense, 362
 American Structuralism, 229
 amplitude
 of a signal, 529
 RMS, 532
 anaphor, 417
 anaphora, 417
 anaphoricity detector, 426
 anchor texts, 437, 481
 anchors in regular expressions, 5, 27
 answer type, 468
 answer type taxonomy, 468
 antecedent, 417
 antonym, 357
 any-of, 68
 AP, 212
 Apple AIFF, 531
 approximant sound, 523
 approximate randomization, 70
 Arabic, 518
 Egyptian, 539
 Aramaic, 518
 ARC, 484
 arc eager, 287
 arc standard, 280
 argumentation mining, 459
 argumentation schemes, 460
 argumentative relations, 459
 argumentative zoning, 461
 Aristotle, 143, 315
 arity, 312
 ARPAbet, 542
 article (part-of-speech), 145
 articulatory phonetics, 520, 520
 aspect, 315
 ASR
 confidence, 509
 association, 96
 ATIS, 203
 corpus, 206, 209
 ATN, 391
 ATRANS, 390
 attachment ambiguity, 233
 attention mechanism, 199
 Attribution (as coherence relation), 446
 AU file, 531
 augmentative communication, 31
 authorship attribution, 56
 autoregressive generation, 177
 auxiliary verb, 146
 B³, 436
 BabelNet, 367
 backoff
 in smoothing, 44
 backprop, 134
 Backpropagation Through Time, 175
 backtrace, 557
 in minimum edit distance, 25
 Backus-Naur Form, 205
 backward chaining, 310
 backward composition, 225
 backward probability, 558
 backward-looking center, 453
 bag of words, 58, 59
 bag-of-words, 58
 barge-in, 512
 baseline
 most frequent sense, 363
 take the first sense, 363
 basic emotions, 396
 batch training, 86
 Bayes' rule, 58
 dropping denominator, 59, 152
 Bayesian inference, 58
 BDI, 516
 Beam search, 288
 beam search, 158, 196
 beam width, 158, 288
 bear pitch accent, 525
 Berkeley Restaurant Project, 34
 Bernoulli naïve Bayes, 72
 BERT
 for affect, 411
 for coreference, 433
 best-worst scaling, 399
 bi-LSTM, 327
 bias term, 77, 124
 bidirectional RNN, 182
 bigram, 32
 bilabial, 521
 binary branching, 222
 binary NB, 63
 binary tree, 222
 bitexts, 193
 bits for measuring entropy, 50
 Bloom filters, 49
 BNF (Backus-Naur Form), 205
 bootstrap, 71
 bootstrap algorithm, 71
 bootstrap test, 70
 bootstrapping, 70
 in IE, 337
 bound pronoun, 419

- boundary tones, 528
 BPE, 18
 bracketed notation, 206
 break index, 528
 bridging inference, 421
 British National Corpus (BNC)
 POS tags for phrases, 147
 Brown, 147
 Brown corpus, 11
 original tagging of, 166
 byte-pair encoding, 18
 canonical form, 300
 capitalization
 for unknown words, 159
 capture group, 10
 cardinal number, 212
 cascade, 22
 regular expression in
 Eliza, 10
 case
 sensitivity in regular
 expression search, 3
 case folding, 21
 case frame, 375, 391
 cataphora, 419
 categorial grammar, 223, 223
 CD (conceptual dependency), 390
 CELEX, 539
 Centering Theory, 444, 452
 centroid, 107
 CFG, *see* context-free
 grammar
 chain rule, 91, 135
 channels in stored
 waveforms, 531
 character embeddings, 330
 Charniak parser, 258
 chatbots, 2, 491
 Chinese
 characters, 518
 Chomsky normal form, 222, 251
 Chomsky-adjunction, 223
 chunking, 240, 241
 CIRCUS, 352
 citation form, 95
Citizen Kane, 443
 CKY algorithm, 232
 probabilistic, 251
 claims, 459
 clarification questions, 511
 class-based n-gram, 54
 clause, 210
 clefts, 422
 clitic, 16
 origin of term, 143
 closed class, 144
 closed vocabulary, 41
 closure, stop, 522
 cluster, 417
 clustering
 in word sense
 disambiguation, 371
 CNF, *see* Chomsky normal
 form
 coarse senses, 371
 cochlea, 537
 Cocke-Kasami-Younger
 algorithm, *see* CKY
 coda, syllable, 524
 code switching, 13
 coherence, 443
 entity-based, 452
 relations, 445
 cohesion
 lexical, 444, 457
 Collins parser, 258
 collocation, 365
 combinatorial categorial
 grammar, 223
 commissive speech act, 489
 common ground, 489, 516
 common nouns, 144
 complement, 215, 215
 complementizer, 145
 completeness in FOL, 311
 componential analysis, 389
 compression, 530
 Computational Grammar
 Coder (CGC), 166
 computational semantics, 298
 concatenation, 27
 conceptual dependency, 390
 concordance, semantic, 362
 concrete word, 398
 conditional independence, 271
 confidence
 ASR, 509
 in relation extraction, 338
 confidence values, 338
 configuration, 278
 conjoined phrase, 216
 conjunction, 145
 conjunctions, 216
 as closed class, 145
 connectionist, 142
 connotation frame, 412
 connotation frames, 392
 connotations, 97, 395
 consistent, 247
 consonant, 520
 constants in FOL, 305
 constative speech act, 489
 constituency, 204
 evidence for, 204
 constituent, 204
 Constraint Grammar, 296
 Construction Grammar, 229
 content planning, 509
 context embedding, 114
 context-free grammar, 203,
 204, 207, 228
 Chomsky normal form,
 222
 invention of, 229
 multiplying probabilities,
 248, 271
 non-terminal symbol,
 205
 productions, 205
 rules, 205
 terminal symbol, 205
 weak and strong
 equivalence, 222
 contingency table, 66
 continuation rise, 528
 conversation, 487
 conversation analysis, 515
 conversational agents, 487
 conversational analysis, 489
 conversational implicature, 491
 convex, 82
 coordinate noun phrase, 216
 coordination ambiguity, 234, 256
 copula, 146
 corefer, 416
 coreference chain, 417
 coreference resolution, 417
 gender agreement, 423
 Hobbs tree search
 algorithm, 440
 number agreement, 422
 person agreement, 423
 recency preferences, 423
 selectional restrictions,
 424
 syntactic (“binding”)
 constraints, 423
 verb semantics, 424
 coronal sound, 522
 corpora, 11
 corpus, 11
 ATIS, 206
 BNC, 147
 Brown, 11, 166
 CASS phonetic of
 Mandarin, 542
 Kiel of German, 542
 LOB, 166
 regular expression
 searching inside, 3
 Switchboard, 11, 495,
 530, 531, 541
 Switchboard phonetic,
 541
 TimeBank, 347
 TIMIT, 541
 correction act detection,
 506
 cosine
 as a similarity metric,
 104
 cost function, 80
 count nouns, 144
 counters, 27
 counts
 treating low as zero, 162
 CRF, 164
 cross-brackets, 270
 cross-entropy, 51
 cross-entropy loss, 81, 133
 cross-validation, 69
 10-fold, 69
 crowdsourcing, 398
 currying, 308
 cycles in a wave, 529
 cycles per second, 529
 date
 fully qualified, 345
 normalization, 501
 dative alternation, 376
 debiasing, 118
 decision boundary, 78, 127
 decision tree
 use in WSD, 371
 declarative sentence
 structure, 209
 decoder, 555
 decoding, 152, 555
 Viterbi, 152, 555
 deduction
 in FOL, 310
 deep
 neural networks, 123
 deep learning, 123
 deep role, 374
 definite reference, 419
 degree adverb, 145
 deleted interpolation, 157
 delexicalization, 510
 denotation, 302
 dental sound, 522
 dependency
 grammar, 273
 lexical, 256
 dependency tree, 276
 dependent, 274
 derivation
 direct (in a formal
 language), 208
 syntactic, 205, 205, 208,
 208
 description logics, 316
 Det, 205
 determiner, 145, 205, 211
 development test set, 69
 development test set
 (dev-test), 36
 devset, *see* development
 test set (dev-test), 69
 dialogue, 487
 dialogue act
 correction, 506
 dialogue acts, 503
 dialogue manager
 design, 512
 dialogue policy, 507
 dialogue systems, 487
 design, 512
 evaluation, 511
 diathesis alternation, 376
 diff program, 28
 digitization, 530
 dimension, 100
 diphthong, 524
 origin of term, 143
 direct derivation (in a
 formal language),
 208
 directional adverb, 145
 directive speech act, 489

- disambiguation
PCFGs for, 247
role of probabilistic parsing, 246
syntactic, 234
via PCFG, 249
- discount, 42, 44, 45
- discounting, 42
- discourse, 443
segment, 446
- discourse connectives, 447
- discourse deixis, 418
- discourse model, 416
- discourse parsing, 448
- discourse-new, 420
- discourse-old, 420
- discovery procedure, 229
- discriminative model, 76
- disfluency, 11
- disjunction, 27
pipe in regular expressions as, 6
square braces in regular expression as, 4
- dispreferred response, 517
- distance
cosine, 104
- distant supervision, 339
- distributional hypothesis, 94
- distributional similarity, 229
- document frequency, 106
- document vector, 107
- domain, 302
- domination in syntax, 205
- dot product, 77, 103
- dropout, 137
- duration
temporal expression, 343
- dynamic programming, 24
and parsing, 234
forward algorithm as, 552
history, 563
Viterbi as, 153, 555
- E-step (expectation step) in EM, 562
- edge-factored, 289
- edit distance
minimum, 24
- EDU, 446
- Elaboration (as coherence relation), 445
- ELIZA, 2
implementation, 10
sample conversation, 10
- Elman Networks, 170
- ELMo
for affect, 411
for coreference, 433
- EM
Baum-Welch as, 558
E-step, 562
for deleted interpolation, 45
- inside-outside in parsing, 253
M-step, 562
- embedded verb, 213
- embeddings, 99
character, 330
cosine for similarity, 103
GloVe, 111
skip-gram, learning, 113
sparse, 103
tf-idf, 105
word2vec, 111
- emission probabilities, 150, 549
- EmoLex, 397
- emotion, 395
- emphatic accent, 525
- empty category, 210
- encoder-decoder, 194
- end-to-end training, 181
- endpointing, 488
- English
simplified grammar rules, 206
- entity grid, 454
- entity linking, 417, 436, 478
- entity-based coherence, 452
- entropy, 50
and perplexity, 50
cross-entropy, 51
per-word, 51
rate, 51
relative, 387
- error backpropagation, 134
- ethos, 459
- Euclidean distance
in L2 regularization, 87
- Eugene Onegin*, 53, 563
- evalb, 270
- evaluating parsers, 268
- evaluation
10-fold cross-validation, 69
comparing models, 38
cross-validation, 69
development test set, 36, 69
devset, 69
devset or development test set, 36
- dialogue systems, 511
- extrinsic, 36
- most frequent class
baseline, 148
- named entity recognition, 331
- of n-gram, 36
- of n-grams via perplexity, 37
- pseudoword, 388
- relation extraction, 342
- test set, 36
training on the test set, 36
training set, 36
- unsupervised WSD, 369
- WSD systems, 363
- event coreference, 418
- Event extraction, 325
- event extraction, 346
- event variable, 312
- events
representation of, 311
- Evidence (as coherence relation), 445
- evoking a referent, 416
- existential there, 146
- expansion, 206, 209
- expectation step, 253
- expectation step in EM, 562
- Expectation-Maximization, *see* EM
- expletive, 422
- explicit confirmation, 508
- expressiveness, of a meaning representation, 301
- extrapolation, 422
- extrinsic evaluation, 36
- F (for F-measure), 67, 243, 270
- F-measure, 67, 243, 243, 270
- F-measure
in NER, 331
- F0, 532
- factoid question, 466
- false negatives, 7
- false positives, 7
- fasttext, 121
- FASTUS, 350
- feature cutoff, 162
- feature interactions, 79
- feature selection
information gain, 73
- feature template, 285
- feature templates, 80
part-of-speech tagging, 161
- Federalist papers, 72
- feedforward network, 129
- file format, .wav, 531
- filled pause, 11
- filler, 11
- final fall, 527
- First Order Logic, *see* FOL
- first-order co-occurrence, 116
- flap (phonetic), 523
- focus, 480
- FOL, 298, 304
 \exists (there exists), 306
 \forall (for all), 306
 \implies (implies), 309
 \wedge (and), 306, 309
 \neg (not), 306, 309
 \vee (or), 309
and verifiability, 304
constants, 305
expressiveness of, 301, 304
- functions, 305
inference in, 304
- terms, 305
- variables, 305
- fold (in cross-validation), 69
- food in NLP
ice cream, 550
- forget gate, 184
- formal language, 207
- formant, 537
- forward algorithm, 552, 553
- FORWARD ALGORITHM, 554
- forward chaining, 310
- forward composition, 225
- forward trellis, 552
- forward-backward
algorithm, 558, 563
backward probability in, 558
relation to inside-outside, 253
- FORWARD-BACKWARD ALGORITHM, 562
- forward-looking centers, 453
- Fosler, E., *see* Fosler-Lussier, E.
- fragment of word, 11
- frame
semantic, 379
- frame elements, 379
- FrameNet, 379
- frames, 498
- free word order, 273
- Freebase, 334
- frequency
of a signal, 529
- fricative sound, 522
- Frump, 352
- fully qualified date
expressions, 345
- fully-connected, 129
- function word, 144, 165
- functional grammar, 230
- functions in FOL, 305
- fundamental frequency, 532
- gaussian
prior on weights, 88
- gazetteer, 329
- General Inquirer, 64, 397
- generalize, 87
- generalized semantic role, 376
- generation
of sentences to test a CFG grammar, 206
template-based, 502
- generative grammar, 207
- generative lexicon, 371
- generative model, 76
- generative syntax, 230
- generator, 205
- generics, 422
- genitive NP, 231
- German, 539
- gerundive postmodifier, 212
- Gilbert and Sullivan, 325
- given-new, 421
- gloss, 359

- glosses, 355
 Glottal, 522
 glottal stop, 522
 glottis, 520
 Godzilla, speaker as, 385
 gold labels, 66
 Good-Turing, 46
 government and binding, 229
 gradient, 83
 Grammar
 Constraint, 296
 Construction, 229
 Government and Binding, 229
 Head-Driven Phrase Structure (HPSG), 220, 229
 Lexical-Functional (LFG), 229
 Link, 296
 Probabilistic Tree Adjoining, 272
 Tree Adjoining, 230
 grammar
 binary branching, 222
 categorial, 223, 223
 CCG, 223
 checking, 232
 combinatory categorial, 223
 equivalence, 222
 generative, 207
 strong equivalence, 222
 weak equivalence, 222
 Grammar Rock, 143
 grammatical function, 274
 grammatical relation, 274
 grammatical sentences, 207
 greedy, 162
 greedy RE patterns, 7
 Greek, 518
 greeting, 146
 grep, 3, 3, 27
 Gricean maxims, 491
 grounding, 489
 GUS, 498
- H* pitch accent, 528
 Hamilton, Alexander, 72
 hanzi, 16
 harmonic, 539
 harmonic mean, 67
 Hays, D., 296
 head, 220, 274
 finding, 220
 in lexicalized grammar, 258
 tag, 259
 head tag, 259
 Head-Driven Phrase Structure Grammar (HPSG), 220, 229
 Heaps' Law, 12
 Hebrew, 518
 held out, 36
 held-out, 45
 Herdan's Law, 12
- hertz as unit of measure, 529
 hidden, 150, 549
 hidden layer, 129
 as representation of input, 130
 hidden units, 129
 HMM, 150, 549
 deleted interpolation, 157
 formal definition of, 150, 549
 initial distribution, 150, 549
 observation likelihood, 150, 549
 observations, 150, 549
 simplifying assumptions for POS tagging, 152
 states, 150, 549
 transition probabilities, 150, 549
 trigram POS tagging, 155
- Hobbs algorithm, 440
 Hobbs tree search algorithm for pronoun resolution, 440
 holonym, 358
 homonymy, 354
 Hungarian part-of-speech tagging, 164
 hyperarticulation, 506
 hypernym, 334, 357
 lexico-syntactic patterns for, 334
- hyperparameters, 137
 hyponym, 357
 Hz as unit of measure, 529
- IBM, 53
 IBM Thomas J. Watson Research Center, 53
 ice cream, 550
 idf, 106
 idf term weighting, 106
 if then reasoning in FOL, 310
 immediately dominates, 205
 imperative sentence structure, 209
 implicature, 491
 implicit argument, 392
 implicit confirmation, 508
 implied hierarchy in description logics, 320
 indefinite article, 211
 indefinite reference, 419
 inference, 301
 in FOL, 310
 inference-based learning, 294
 infinitives, 215
 infoboxes, 333
 information structure, 420
 status, 420
- information extraction (IE), 325
 bootstrapping, 337
 partial parsing for, 240
 information gain, 73
 for feature selection, 73
 Information retrieval, 102
 initiative, 490
 inner ear, 537
 inner product, 103
 inside-outside algorithm, 253, 271
 instance checking, 319
 Institutional Review Board, 514
 intensity of sound, 533
 intent determination, 500
 intercept, 77
 interjection, 146
 intermediate phrase, 527
 in ToBI, 528
 internal rule in a CFG parse, 259
 International Phonetic Alphabet, 519, 542
 interpersonal stance, 409
 Interpolated Kneser-Ney discounting, 46, 48
 interpolation in smoothing, 44
 interpretable, 91
 interpretation, 302
 intonation phrases, 527
 intransitive verbs, 215
 intrinsic evaluation, 36
 IOB, 241, 328, 504
 IOB tagging for NER, 328
 for temporal expressions, 343
 IPA, 519, 542
 IR
 idf term weighting, 106
 vector space model, 100
- IRB, 514
 IS-A, 358
 is-a, 334
 ISO 8601, 344
 iSRL, 392
- Japanese, 518, 539
 Jay, John, 72
 joint intention, 516
 joint probability, 248
- Katz backoff, 45
 KBP, 352
 KenLM, 49, 54
 KL divergence, 387
 KL-ONE, 322
 Kleene *, 5
 sneakiness of matching zero things, 5
 Kleene +, 5
 Kneser-Ney discounting, 46
 knowledge base, 300
 knowledge claim, 461
 knowledge-based, 365
- Korean, 539
 KRL, 322
 Kullback-Leibler divergence, 387
 L* pitch accent, 528
 L+H* pitch accent, 528
 L1 regularization, 88
 L2 regularization, 87
 label bias, 163
 labeled precision, 270
 labeled recall, 270
 labial place of articulation, 521
 labiodental consonants, 521
 lambda notation, 308
 language ID, 64
 language id, 56
 language model, 31
 PCFG, 250
 Laplace smoothing, 42
 Laplace smoothing-for PMI, 110
 larynx, 520
 lasso regression, 88
 latent semantic analysis, 120
 lateral sound, 523
 LDC, 16, 253
 learning rate, 83
 lemma, 12, 95
 versus wordform, 12
 lemmatization, 3
 Lesk algorithm, 365
 Simplified, 365
 Levenshtein distance, 23
 lexical category, 205
 cohesion, 444, 457
 database, 359
 dependency, 256
 head, 271
 semantics, 95
 stress, 525
 trigger, in IE, 343
 lexical answer type, 480
 lexical dependency, 256
 lexical rule in a CFG parse, 259
 lexical sample task in WSD, 362
 Lexical-Functional Grammar (LFG), 229
 lexicalized grammar, 258
 lexico-syntactic pattern, 334
 lexicon, 205
 likelihood, 59
 linear classifiers, 60
 linear interpolation for n-grams, 45
 linearly separable, 127
 Linguistic Data Consortium, 16, 253
 Linguistic Discourse model, 462
 Link Grammar, 296

- List (as coherence relation), **446**
LIWC, **64, 398**
LM, **31**
LOB corpus, **166**
locative, **145**
locative adverb, **145**
log
 why used for
 probabilities, **35**
 why used to compress
 speech, **531**
log likelihood ratio, **406**
log odds ratio, **406**
log probabilities, **35, 35**
logical connectives, **306**
logical vocabulary, **301**
logistic function, **77**
logistic regression, **75**
 conditional maximum
 likelihood
 estimation, **81**
 Gaussian priors, **88**
 learning in, **80**
 regularization, **88**
 relation to neural
 networks, **131**
logos, **459**
Long short-term memory, **184**
long-distance dependency, **217**
 traces in the Penn
 Treebank, **217**
 wh-questions, **210**
lookahead in RE, **11**
loss, **80**
loudness, **534**
LSI, *see* latent semantic
 analysis
LSTM, **352**
 for NER, **330**
 for SRL, **383**
LUNAR, **485**
Lunar, **322**
- M-step (maximization step)**
 in EM, **562**
machine learning
 for NER, **331**
 textbooks, **73, 93**
macroaveraging, **68**
Madison, James, **72**
Mandarin, **539**
Manhattan distance
 in L1 regularization, **88**
manner adverb, **145**
manner of articulation, **522**
marker passing for WSD, **370**
Markov, **33**
 assumption, **33**
Markov assumption, **149, 548**
Markov chain, **53, 149, 548**
 formal definition of, **150, 549**
 initial distribution, **150, 549**
 N-gram as, **150, 549**
 states, **150, 549**
 transition probabilities, **150, 549**
Markov model, **33**
 formal definition of, **150, 549**
 history, **53**
Marx, G., **232**
mass nouns, **144**
maxent, **93**
maxim, Gricean, **491**
maximization step, **253**
maximization step in EM, **562**
maximum entropy, **93**
maximum matching, **19**
maximum spanning tree, **290**
MaxMatch, **19**
MCTest, **484**
mean reciprocal rank, **483**
meaning representation, **298**
 as set of symbols, **299**
 early uses, **322**
 languages, **299**
mechanical indexing, **120**
mel
 scale, **533**
MEMM, **160**
 compared to HMM, **160**
 inference (decoding), **163**
 learning, **163**
 Viterbi decoding, **163**
mention detection, **425**
mention-pair, **428**
mentions, **416**
meronym, **358**
meronymy, **358**
**MeSH (Medical Subject
 Headings)**, **57, 362**
**Message Understanding
 Conference**, **350**
metarule, **216**
metonymy, **358, 442**
Micro-Planner, **322**
microaveraging, **68**
Microsoft .wav format, **531**
mini-batch, **86**
minimum edit distance, **23, 24, 153, 555**
 example of, **26**
MINIMUM EDIT DISTANCE, **26**
MLE
 for n-grams, **33**
 for n-grams, intuition, **34**
MLP, **129**
modal verb, **146**
model, **301**
 modified Kneser-Ney, **48**
 modus ponens, **310**
 Montague semantics, **322**
morpheme, **21**
Moses, Michelangelo statue
 of, **487**
most frequent sense, **363**
MRR, **483**
MUC, **350, 352**
MUC F-measure, **436**
multi-label classification, **68**
multi-layer perceptrons, **129**
multinomial classification, **68**
**multinomial logistic
 regression**, **89**
multinomial naive Bayes, **58**
**multinomial naive Bayes
 classifier**, **58**
N-gram
 as Markov chain, **150, 549**
n-gram, **31, 33**
 absolute discounting, **46**
 add-one smoothing, **42**
 as approximation, **32**
 as generators, **39**
 equation for, **33**
 example of, **35**
 for Shakespeare, **39**
 history of, **53**
 interpolation, **44**
 Katz backoff, **45**
KenLM, **49, 54**
 Kneser-Ney discounting, **46**
 logprobs in, **35**
 normalizing, **34**
 parameter estimation, **34**
 sensitivity to corpus, **38**
 smoothing, **42**
SRILM, **54**
 test set, **36**
 training set, **36**
 unknown words, **41**
n-gram
 tiling, **472**
naive Bayes
 multinomial, **58**
 simplifying assumptions, **59**
naive Bayes assumption, **59**
naive Bayes classifier
 use in text categorization, **58**
named entity, **326**
 list of types, **327**
 recognition, **325, 327**
named entity recognition, **179**
names
 and gazetteers, **329**
 census lists, **329**
NarrativeQA, **483**
nasal sound, **520, 522**
nasal tract, **520**
negative log likelihood loss, **133**
negative part-of-speech, **146**
neo-Davidsonian, **312**
NER, **325**
neural networks
 relation to logistic
 regression, **131**
newline character, **9**
noisy-or, **338**
NomBank, **378**
Nominal, **205**
non-capturing group, **10**
non-finite postmodifier, **212**
non-greedy, **7**
non-logical vocabulary, **301**
non-terminal symbols, **205, 206**
normal form, **222, 222**
normalization
 dates, **501**
 temporal, **344**
 word, **21**
**normalization of
 probabilities**, **33**
normalized, **326**
normalizing, **131**
noun, **144**
 abstract, **144, 211**
 common, **144**
 count, **144**
 days of the week coded
 as, **145**
 mass, **144, 211**
 proper, **144**
noun phrase, **204**
 constituents, **205**
NP, **205, 206**
NP attachment, **255**
nuclear accent, **525**
nucleus, **445**
nucleus of syllable, **524**
null hypothesis, **70**
numerals
 as closed class, **145**
Nyquist frequency, **531**
- o**, **521**
**object, syntactic
 frequency of pronouns**
 as, **254**
observation bias, **163**
**observation likelihood
 role in forward**, **553**
 role in Viterbi, **154, 556**
**old information, and word
 order**, **254**
one-hot vector, **139**
one-of, **68**
onset, syllable, **524**
ontology, **316**
OntoNotes, **371**
**OOV (out of vocabulary)
 words**, **41**
OOV rate, **41**
open class, **144**
**open information
 extraction**, **340**
open vocabulary system

- unknown words in, 41
 operation list, 23
 operator precedence, 6, 6
 optionality
 of determiners, 211
 use of ? in regular
 expressions for, 4
 oral tract, 520
 ordinal number, 212
 orthography
 opaque, 519
 transparent, 519
 output gate, 185
 overfitting, 87
 palatal sound, 522
 palate, 522
 palato-alveolar sound, 522
 parallel distributed
 processing, 142
 parent annotation, 257
 parse tree, 205, 208
 parsed corpus, 271
 parsing
 ambiguity, 232
 chunking, 240
 CKY, 235, 251
 CYK, *see* CKY
 evaluation, 268
 history, 244
 partial, 240
 probabilistic CKY, 251
 relation to grammars,
 209
 shallow, 240
 syntactic, 232
 well-formed substring
 table, 244
 part-of-speech
 adjective, 144
 adverb, 144
 as used in CFG, 205
 closed class, 144, 145
 greeting, 146
 interjection, 146
 negative, 146
 noun, 144
 open class, 144
 particle, 145
 subtle distinction
 between verb and
 noun, 144
 usefulness of, 143
 verb, 144
 part-of-speech tagger
 PARTS, 166
 TAGGIT, 166
 Part-of-speech tagging, 148
 part-of-speech tagging
 ambiguity and, 148
 amount of ambiguity in
 Brown corpus, 148
 and morphological
 analysis, 164
 capitalization, 159
 feature templates, 161
 for phrases, 147
 history of, 166
 Hungarian, 164
 Stanford tagger, 164
 state of the art, 149
 Turkish, 164
 unknown words, 159
 part-whole, 358
 partial parsing, 240
 particle, 145
 PARTS tagger, 166
 parts of speech, 143
 passage retrieval, 469
 passages, 469
 pathos, 459
 pattern, regular expression,
 3
 PCFG, 246
 for disambiguation, 247
 lack of lexical sensitivity,
 254
 lexicalized, 271
 parse probability, 248
 poor independence
 assumption, 254
 rule probabilities, 247
 use in language
 modeling, 250
 PCM (Pulse Code
 Modulation), 531
 PDP, 142
 PDTB, 447
 Penn Discourse TreeBank,
 447
 Penn Treebank, 217
 for statistical parsing,
 253
 POS tags for phrases,
 147
 tagging accuracy, 149
 tagset, 146, 146
 Penn Treebank
 tokenization, 16
 per-word entropy, 51
 perceptron, 126
 period of a wave, 529
 perplexity, 37, 52
 as weighted average
 branching factor, 37
 defined via
 cross-entropy, 52
 personal pronoun, 145
 personality, 409
 persuasion, 460
 phone, 518, 542
 phoneme, 542
 phonetics, 518
 articulatory, 520, 520
 phonotactics, 525
 phrasal verb, 145
 phrase-structure grammar,
 205, 229
 PII, 495
 pipe, 6
The Pirates of Penzance,
 325
 pitch, 533
 pitch accent, 525
 ToBI, 528
 pitch extraction, 534
 pitch track, 532
 planning
 and speech acts, 516
 shared plans, 516
 pleonastic, 422
 plosive sound, 522
 plural, 211
 Pointwise mutual
 information, 108
 politeness marker, 146
 Porter stemmer, 21
 POS, 143
 possessive NP, 231
 possessive pronoun, 145
 postdeterminer, 212
 postmodifier, 212
 postposed constructions,
 204
 Potts diagram, 405
 power of a signal, 533
 PP, 206
 PPMI, 108
 praat, 534, 535, 542
 precedence, 6
 precedence, operator, 6
 Precision, 67
 precision, 243
 in NER, 331
 predeterminer, 213
 predicate, 215
 predicate-argument
 relations, 215
 preference semantics, 370
 premises, 459
 preposed constructions, 204
 prepositional phrase, 212
 attachment, 255
 constituency, 206
 preposing, 204
 prepositions, 145
 as closed class, 145
 presquences, 490
 pretraining, 138
 primitive decomposition,
 389
 principle of contrast, 96
 prior probability, 59
 probabilistic CKY
 algorithm, 251, 251
 probabilistic parsing, 251
 productions, 205
 progressive prompting, 508
 projection layer, 140
 Prolog, 310
 prominence, phonetic, 526
 prominent word, 525
 prompts, 502
 pronoun, 145
 and old information, 254
 as closed class, 145
 bound, 419
 demonstrative, 420
 non-binary, 423
 personal, 145
 possessive, 145
 wh-, 145
 pronunciation dictionary,
 539
 CELEX, 539
 CMU, 540
 PRONLEX, 541
 PropBank, 377
 proper noun, 144
 propositional meaning, 96
 prosodic phrasing, 527
 prosody, 526
 accented syllables, 525
 reduced vowels, 526
 PROTO-AGENT, 376
 PROTO-PATIENT, 376
 pseudoword, 388
 PTAG, 272
 PTRANS, 390
 punctuation
 for numbers
 cross-linguistically,
 16
 for sentence
 segmentation, 22
 part-of-speech tags, 146
 stripping before
 part-of-speech
 tagging, 148
 tokenization, 15
 treated as words, 11
 treated as words in LM,
 40
 QuAC, 483
 qualia structure, 371
 quantifier
 as part of speech, 212
 semantics, 306
 quantization, 531
 query
 reformulation in QA, 468
 question
 classification, 468
 factoid, 466
 rise, 527
 question answering
 evaluation, 483
 factoid questions, 466
 query reformulation in,
 468
 range, regular expression, 4
 rapid reprompting, 509
 rarefaction, 530
 RDF, 334
 RDF triple, 334
 RE
 regular expression, 3
 reading comprehension,
 472
 Reason (as coherence
 relation), 445
 Recall, 67
 recall, 243
 in NER, 331
 recipe
 meaning of, 298
 recurrent neural networks,
 170
 reduced vowels, 526
 reduction, phonetic, 526

- reference
 bound pronouns, 419
 cataphora, 419
 definite, 419
 generics, 422
 indefinite, 419
 reference point, 314
 referent, 416
 accessing of, 416
 evoking of, 416
 reflexive, 423
 register in RE, 10
 regression
 lasso, 88
 ridge, 88
 regular expression, 3, 27
 substitutions, 9
 regularization, 87
 rejection
 conversation act, 508
 relatedness, 96
 relation extraction, 325
 relative
 temporal expression, 343
 relative entropy, 387
 relative frequency, 34
 relative pronoun, 213
 release, stop, 522
 relevance, 491
 relexicalize, 510
 ReLU, 125
 reporting events, 347
 representation learning, 94
 resolution for inference,
 311
 resolve, 148
 response generation, 496
 restrictive grammar, 502
 restrictive relative clause,
 213
 retrofitting, 368
 ReVerb, 340
 reversives, 357
 rewrite, 205
 Rhetorical Structure
 Theory, *see* RST
 rhyme, syllable, 524
 Riau Indonesian, 144
 ridge regression, 88
 rime
 syllable, 524
 RMS amplitude, 532
 role-filler extraction, 349
 Rosebud, sled named, 443
 rounded vowels, 524
 row vector, 102
 RST, 445
 TreeBank, 447, 462
 rules
 context-free, 205
 context-free, expansion,
 205, 209
 context-free, sample, 206

S as start symbol in CFG,
 205
 SAE, 13
- salience, in discourse
 model, 421
 SAM phonetic
 transcription, 539
 sampling
 of analog waveform, 530
 rate, 530
 used in clustering, 369
 satellite, 445
 saturated, 126
 Schönfinkelization, 308
 "Schoolhouse Rock", 143
 schwa, 526
 SCISOR, 352
 scilite package, 28
 script
 Schankian, 379
 scripts, 349
 SDRT (Segmented
 Discourse
 Representation
 Theory), 462
 second-order
 co-occurrence, 116
 secondary stress, 526
 seed pattern in IE, 337
 seed tuples, 337
 segmentation
 maximum matching, 19
 sentence, 22
 word, 15
 selectional association, 388
 selectional preference
 strength, 387
 selectional preferences
 pseudowords for
 evaluation, 388
 selectional restriction, 384
 representing with events,
 385
 violations in WSD, 387
 semantic concordance, 362
 semantic drift in IE, 338
 semantic feature, 120
 semantic field, 97
 semantic frame, 97
 semantic grammars, 501
 semantic network
 for word sense
 disambiguation, 370
 semantic networks
 origins, 322
 semantic parsing, 298
 semantic relations in IE,
 332
 table, 333
 semantic role, 374, 374,
 376
 Semantic role labeling, 380
 semantics
 lexical, 95
 semivowel, 521
 sense
 word, 354, 355
 sentence
 segmentation, 22
 sentence realization, 509
 sentence segmentation, 3
- sentence selection, 473
 SentencePiece, 20
 sentential complements,
 214
 sentiment, 97
 origin of term, 414
 sentiment analysis, 56
 sentiment lexicons, 64
 SentiWordNet, 403
 sequence model, 149
 SGNS, 111
 Shakespeare
 n-gram approximations
 to, 39
 shallow discourse parsing,
 451
 shallow parse, 240
 shared plans, 516
 shift-reduce parsing, 278
 SHRDLU, 322
 sibilant sound, 523
 side sequence, 490
 sigmoid, 77, 124
 similarity, 96
 Simplified Lesk, 365
 singleton, 417
 singular they, 423
 skip-gram, 111
 slot filling, 352, 500
 slots, 498
 smoothing, 42, 42
 absolute discounting, 46
 add-one, 42
 discounting, 42
 for HMM POS tagging,
 157
 interpolation, 44
 Katz backoff, 45
 Kneser-Ney discounting,
 46
 Laplace, 42
 linear interpolation, 45
 snippets, 469
 softmax, 89, 131
 source-filter model, 539
 spam detection, 56, 64
 span, 471
 Spanish, 539
 spectrogram, 537
 spectrum, 535
 speech
 telephone bandwidth,
 531
 wideband, 531
 speech acts, 489
 spelling correction
 use of n-grams in, 30
 split, 256
 split and merge, 258
 split-half reliability, 400
 SQuAD, 472, 483
 SRILM, 54
 SRL, 380
 Stacked RNNs, 181
 Stanford tagger, 164
 start symbol, 205
 state
- semantic representation
 of, 311
 stationary stochastic
 process, 51
 statistical parsing, 251
 stative expressions, 315
 stem, 21
 Stemming, 3
 stemming, 21
 stop (consonant), 522
 stop words, 61
 stress
 lexical, 525
 secondary, 526
 strong equivalence of
 grammars, 222
 structural ambiguity, 232
 structure prediction, 179
 structured polysemy, 358
 stupid backoff, 49
 subcategorization
 tags for, 215
 subcategorization frame,
 215
 examples, 215
 subcategorize for, 215
 subdialogue, 490
 subject, syntactic
 frequency of pronouns
 as, 254
 in wh-questions, 210
 subjectivity, 394, 414
 substitutability, 229
 substitution in TAG, 230
 substitution operator
 (regular
 expressions), 9
 subsumption, 317, 319
 subword, 17
 superordinate, 358
 supersenses, 360
 Supertagging, 264
 supervised machine
 learning, 57
 suprasegmental, 526
 SVD, 120
 Switchboard, 147
 Transcription Project,
 541
 Switchboard Corpus, 11,
 495, 530, 531, 541
 syllabification, 525
 syllable, 524
 accented, 525
 coda, 524
 nucleus, 524
 onset, 524
 prominent, 525
 rhyme, 524
 rime, 524
 synonyms, 96, 357
 synset, 359
 syntactic categories, 143
 syntactic disambiguation,
 234
 syntactic movement, 217
 syntax, 203
 origin of term, 143

- TAG, **230**, 272
 TAGGIT, 166
 tagset
 difference between Penn Treebank and Brown, 147
 history of Penn Treebank, 147
 Penn Treebank, **146**, **146**
 table of Penn Treebank tags, 146
 tanh, **125**
 tap (phonetic), **523**
 target embedding, **114**
 Tay, **514**
 TBox, **316**
 technai, 143
 telephone-bandwidth speech, **531**
 telic eventualities, **316**
 template filling, **326**, **349**
 template recognition, **349**
 template, in IE, 349
 template-based generation, **502**
 temporal adverb, **145**
 temporal anchor, **346**
 temporal expression
 absolute, 343
 metaphor for, 315
 recognition, 326
 relative, 343
 temporal expressions, **326**
 temporal logic, **312**
 temporal normalization, **344**
 temporal reasoning, 323
 tense logic, **312**
 term
 clustering, 370, 371
 in FOL, **305**
 term frequency, **105**
 term-document matrix, **100**
 term-term matrix, **102**
 terminal symbol, **205**
 terminology
 in description logics, 316
 test set, **36**
 development, **36**
 how to choose, 36
 text categorization, **56**
 bag of words assumption, 58
 naive Bayes approach, 58
 unknown words, 61
 text normalization, **2**
 part-of-speech tagging, 147
 TextTiling, **457**
 tf-idf, **107**
 thematic grid, **375**
 thematic role, **374**
 and diathesis alternation, **376**
 examples of, 374
 problems, 376
 theme, **374**
 theme, as thematic role, **374**
- there, existential in English, **146**
 thesaurus, 370
 tier
 of ToBI transcript, 528
 time, representation of, 312
 time-aligned transcription, **541**
 TimeBank, **347**
 TIMIT, 541
 ToBI, **528**
 boundary tones, 528
 tiers, 528
 tokenization, **2**
 maximum matching, **19**
 sentence, 22
 word, **15**
 tokens, word, **12**
 topic (information structure), 254
 topic models, **97**
 trace, 210, **217**
 trachea, **520**
 training oracle, **283**
 training set, **36**
 cross-validation, 69
 how to choose, 36
 transcription
 time-aligned, **541**
 Transformations and
 Discourse Analysis Project (TDAP), 166
 transformer
 for NER, 327
 transition probability
 role in forward, 553
 role in Viterbi, 154, 556
 transitive verbs, **215**
 TREC, 485
 Tree Adjoining Grammar (TAG), **230**
 adjunction in, 230
 probabilistic, 272
 substitution in, 230
 treebank, **216**, 253
 trigram, **35**
 truth-conditional semantics, **303**
 tune, **527**
 continuation rise, 528
 Turing test
 Passed in 1972, 495
 Turkish
 part-of-speech tagging, 164
 turn correction ratio, 512
 turns, **488**
 type raising, **225**
 typed dependency structure, **273**
 types
 word, **12**
 ungrammatical sentences, **207**
 unit production, **235**
 unit vector, **104**
- Universal Dependencies, **275**
 Unix, 3
 <UNK>, **41**
 unknown words
 in n-grams, **41**
 in part-of-speech tagging, 159
 in text categorization, 61
 unvoiced sound, **520**
 user-centered design, 512
 utterance, **11**
- vagueness, **300**
 vanishing gradients, **184**
 variable
 existentially quantified, 307
 universally quantified, 307
 variables, **301**
 variables in FOL, **305**
 vector, **100**, **124**
 vector length, **104**
 vector semantics, **94**, **98**
 vector space, **100**
 vector space model, **100**
 velar sound, **522**
 velum, **522**
 verb
 copula, **146**
 modal, **146**
 phrasal, **145**
 verb alternations, **376**
 verb phrase, **206**, **214**
 Verbs, **144**
 verifiability, **299**
 Viterbi algorithm, 24, **153**, **555**
 backtrace in, **557**
 decoding in MEMM, 163
 history of, 563
 VITERBI ALGORITHM, 153, 556
 vocal
 cords, **520**
 folds, **520**
 tract, **520**
 voice user interface, **512**
 voiced sound, **520**
 voiceless sound, **520**
 vowel, **520**
 back, **523**
 front, **523**
 height, **523**
 high, **523**
 low, **523**
 mid, **523**
 reduced, **526**
 rounded, **523**
 VP attachment, **255**
- wavefile format, **531**
 weak equivalence of grammars, 222
 Web Ontology Language, **321**
 WebQuestions, 483
- well-formed substring table, 244
 WFST, **244**
 wh-non-subject-question, **210**
 wh-phrase, **210**, **210**
 wh-pronoun, **145**
 wh-subject-questions, **210**
 wh-word, **210**
 WiC, **367**
 wideband speech, **531**
 wikification, **436**
 WikiQA, 483
 wildcard, regular expression, **5**
 Winograd Schema, **437**
 Wizard-of-Oz system, **513**
 word
 boundary, regular expression notation, 6
 closed class, **144**
 definition of, 11
 fragment, **11**
 function, **144**, 165
 open class, **144**
 punctuation as, 11
 tokens, **12**
 types, **12**
 word normalization, **21**
 word segmentation, **15**, **17**
 word sense, **354**, **355**
 word sense disambiguation, **362**, *see* WSD
 word sense induction, **368**
 word shape, **161**, **329**
 word tokenization, **15**
 word-in-context, **366**
 word-word matrix, **102**
 word2vec, **111**
 wordform, **12**
 and lemma, **95**
 versus lemma, 12
 WordNet, 359, **359**
 wordpiece, **19**
 world knowledge, 298
 WSD, **362**
 AI-oriented efforts, 370
 all-words task, **362**
 bootstrapping, 371
 decision tree approach, 371
 evaluation of, 363
 history, 370
 history of, 371
 lexical sample task, **362**
 neural network approaches, 370
 robust approach, 370
 supervised machine learning, 371
 unsupervised machine learning, 368
 WSI, **368**
 WSJ, **147**
- X-bar schemata, **229**

yes-no questions, **209**
yield, **249**

Yonkers Racetrack, 50

zero anaphor, **420**
zero-width, **11**

zeros, **41**
zeugma, **356**