CHAPTER 24

NATURAL LANGUAGE PROCESSING

In which we see how a computer can use natural language to communicate with humans and learn from what they have written.

About 100,000 years ago, humans learned how to speak, and about 5,000 years ago they learned to write. The complexity and diversity of human language sets *Homo sapiens* apart from all other species. Of course there are other attributes that are uniquely human: no other species wears clothes, creates art, or spends two hours a day on social media in the way that humans do. But when Alan Turing proposed his test for intelligence, he based it on language, not art or haberdashery, perhaps because of its universal scope and because language captures so much of intelligent behavior: a speaker (or writer) has the **goal** of communicating some **knowledge**, then **plans** some language that **represents** the knowledge, and **acts** to achieve the goal. The listener (or reader) **perceives** the language, and **infers** the intended meaning. This type of communication via language has allowed civilization to grow; it is our main means of passing along cultural, legal, scientific, and technological knowledge. There are three primary reasons for computers to do **natural language processing (NLP)**:

Natural language processing (NLP)

- To **communicate** with humans. In many situations it is convenient for humans to use speech to interact with computers, and in most situations it is more convenient to use natural language rather than a formal language such as first-order predicate calculus.
- To **learn**. Humans have written down a lot of knowledge using natural language. Wikipedia alone has 30 million pages of facts such as "Bush babies are small nocturnal primates," whereas there are hardly any sources of facts like this written in formal logic. If we want our system to know a lot, it had better understand natural language.
- To advance the **scientific understanding** of languages and language use, using the tools of AI in conjunction with linguistics, cognitive psychology, and neuroscience.

In this chapter we examine various mathematical models for language, and discuss the tasks that can be achieved using them.

24.1 Language Models

Formal languages, such as first-order logic, are precisely defined, as we saw in Chapter 8. A **grammar** defines the syntax of legal sentences and **semantic rules** define the meaning.

Natural languages, such as English or Chinese, cannot be so neatly characterized:

- Language judgments vary from person to person and time to time. Everyone agrees that "Not to be invited is sad" is a grammatical sentence of English, but people disagree on the grammaticality of "To be not invited is sad."

- Natural language is **ambiguous** ("He saw her duck" can mean either that she owns a waterfowl, or that she made a downwards evasive move) and **vague** ("That's great!" does not specify precisely how great it is, nor what it is).
- The mapping from symbols to objects is not formally defined. In first-order logic, two uses of the symbol "Richard" must refer to the same person, but in natural language two occurrences of the same word or phrase may refer to different things in the world.

If we can't make a definitive Boolean distinction between grammatical and ungrammatical strings, we can at least say how likely or unlikely each one is.

Language model

We define a **language model** as a probability distribution describing the likelihood of any string. Such a model should say that "Do I dare disturb the universe?" has a reasonable probability as a string of English, but "Universe dare the I disturb do?" is extremely unlikely.

With a language model, we can predict what words are likely to come next in a text, and thereby suggest completions for an email or text message. We can compute which alterations to a text would make it more probable, and thereby suggest spelling or grammar corrections. With a pair of models, we can compute the most probable translation of a sentence. With some example question/answer pairs as training data, we can compute the most likely answer to a question. So language models are at the heart of a broad range of natural language tasks. The language modeling task itself also serves as a common benchmark to measure progress in language understanding.

Natural languages are complex, so any language model will be, at best, an approximation. The linguist Edward Sapir said "No language is tyrannically consistent. All grammars leak" (Sapir, 1921). The philosopher Donald Davidson said "there is no such thing as language, not if a language is . . . a clearly defined shared structure" (Davidson, 1986), by which he meant there is no one definitive language model for English in the way that there is for Python 3.8; we all have different models, but we still somehow manage to muddle through and communicate. In this section we cover simplistic language models that are clearly wrong, but still manage to be useful for certain tasks.

24.1.1 The bag-of-words model

Section 12.6.1 explained how a naive Bayes model based on the presence of specific words could reliably classify sentences into categories; for example sentence (1) below is categorized as *business*, and (2) as *weather*.

1. Stocks rallied on Monday, with major indexes gaining 1% as optimism persisted over the first quarter earnings season.
2. Heavy rain continued to pound much of the east coast on Monday, with flood warnings issued in New York City and other locations.

This section revisits the naive Bayes model, casting it as a full language model. That means we don't just want to know what category is most likely for each sentence; we want a joint probability distribution over all sentences and categories. That suggests we should consider *all* the words in the sentence. Given a sentence consisting of the words $w_1, w_2, \ldots w_N$ (which we will write as $w_{1:N}$, as in Chapter 14), the naive Bayes formula (Equation (12.21)) gives us

$$\mathbf{P}(Class \,|\, w_{1:N}) \;=\; \alpha\, \mathbf{P}(Class) \prod_j \mathbf{P}(w_j \,|\, Class)\,.$$

Bag-of-words model

The application of naive Bayes to strings of words is called the **bag-of-words model**. It is a generative model that describes a process for generating a sentence: Imagine that for each category (*business*, *weather*, etc.) we have a bag full of words (you can imagine each word written on a slip of paper inside the bag; the more common the word, the more slips it is duplicated on). To generate text, first select one of the bags and discard the others. Reach into that bag and pull out a word at random; this will be the first word of the sentence. Then put the word back and draw a second word. Repeat until an end-of-sentence indicator (e.g., a period) is drawn.

This model is clearly wrong: it falsely assumes that each word is independent of the others, and therefore it does not generate coherent English sentences. But it does allow us to do classification with good accuracy using the naive Bayes formula: the words "stocks" and "earnings" are clear evidence for the business section, while "rain" and "cloudy" suggest the weather section.

Corpus

We can learn the prior probabilities needed for this model via supervised training on a body or **corpus** of text, where each segment of text is labeled with a class. A corpus typically consists of at least a million words of text, and at least tens of thousands of distinct vocabulary words. Recently we are seeing even larger corpuses being used, such as the 2.5 billion words in Wikipedia or the 14 billion word iWeb corpus scraped from 22 million web pages.

From a corpus we can estimate the prior probability of each category, $\mathbf{P}(Class)$, by counting how common each category is. We can also use counts to estimate the conditional probability of each word given the category, $\mathbf{P}(w_j \mid Class)$. For example, if we've seen 3000 texts and 300 of them were classified as *business*, then we can estimate $P(Class = business) \approx 300/3000 = 0.1$. And if within the *business* category we have seen 100,000 words and the word "stocks" appeared 700 times, then we can estimate $P(stocks \mid Class = business) \approx 700/100{,}000 = 0.007$. Estimation by counting works well when we have high counts (and low variance), but we will see in Section 24.1.4 a better way to estimate probabilities when the counts are low.

Sometimes a different machine learning approach, such as logistic regression, neural networks, or support vector machines, can work even better than naive Bayes. The features of the machine learning model are the words in the vocabulary: "a," "aardvark," ..., "zyzzyva," and the values are the number of times each word appears in the text (or sometimes just a Boolean value indicating whether the word appears or not). That makes the feature vector large and sparse—we might have 100,000 words in the language model, and thus a feature vector of length 100,000, but for a short text almost all the features will be zero.

As we have seen, some machine learning models work better when we do **feature selection**, limiting ourselves to a subset of the words as features. We could drop words that are very rare (and thus have high variance in their predictive powers), as well as words that are common to all classes (such as "the") but don't discriminate between classes. We can also mix other features in with our word-based features; for example if we are classifying email messages we could add features for the sender, the time the message was sent, the words in the subject header, the presence of nonstandard punctuation, the percentage of uppercase letters, whether there is an attachment, and so on.

Note it is not trivial to decide what a *word* is. Is "aren't" one word, or should it be broken up as "aren/'/t" or "are/n't," or something else? The process of dividing a text into a sequence of words is called **tokenization**.

Tokenization

24.1.2 N-gram word models

The bag-of-words model has limitations. For example, the word "quarter" is common in both the *business* and *sports* categories. But the four-word sequence "first quarter earnings report" is common only in *business* and "fourth quarter touchdown passes" is common only in *sports*. We'd like our model to make that distinction. We could tweak the bag-of-words model by treating special phrases like "first-quarter earnings report" as if they were single words, but a more principled approach is to introduce a new model, where each word is dependent on previous words. We can start by making a word dependent on *all* previous words in a sentence:

$$P(w_{1:N}) = \prod_{j=1}^{N} P(w_j \mid w_{1:j-1}).$$

This model is in a sense perfectly "correct" in that it captures all possible interactions between words, but it is not practical: with a vocabulary of 100,000 words and a sentence length of 40, this model would have 10^{200} parameters to estimate. We can compromise with a **Markov chain** model that considers only the dependence between n adjacent words. This is known as an **n-gram model** (from the Greek root *gramma* meaning "written thing"): a sequence of written symbols of length n is called an n-gram, with special cases "unigram" for 1-gram, "bigram" for 2-gram, and "trigram" for 3-gram. In an n-gram model, the probability of each word is dependent only on the $n-1$ previous words; that is:

N-gram model

$$\begin{aligned} P(w_j \mid w_{1:j-1}) &= P(w_j \mid w_{j-n+1:j-1}) \\ P(w_{1:N}) &= \prod_{j=1}^{N} P(w_j \mid w_{j-n+1:j-1}). \end{aligned}$$

N-gram models work well for classifying newspaper sections, as well as for other classification tasks such as **spam detection** (distinguishing spam email from non-spam), **sentiment analysis** (classifying a movie or product review as positive or negative) and **author attribution** (Hemingway has a different style and vocabulary than Faulkner or Shakespeare).

Spam detection

Sentiment analysis

Author attribution

24.1.3 Other n-gram models

An alternative to an n-gram word model is a **character-level model** in which the probability of each character is determined by the $n-1$ previous characters. This approach is helpful for dealing with unknown words, and for languages that tend to run words together, as in the Danish word "Speciallægepraksisplanlægningsstabiliseringsperiode."

Character-level model

Character-level models are well suited for the task of **language identification**: given a text, determine what language it is written in. Even with very short texts such as "Hello, world" or "Wie geht's dir," n-gram letter models can identify the first as English and the second as German, generally achieving accuracy greater than 99%. (Closely related languages such as Swedish and Norwegian are more difficult to distinguish and require longer samples; there, accuracy is in the 95% range.) Character models are good at certain classification tasks, such as deciding that "dextroamphetamine" is a drug name, "Kallenberger" is a person name, and "Plattsburg" is a city name, even if we have never seen these words before.

Language identification

Another possibility is the **skip-gram** model, in which we count words that are near each other, but skip a word (or more) between them. For example, given the French text "je ne comprends pas" the 1-skip-bigrams are "je comprends," and "ne pas." Gathering these

Skip-gram

helps create a better model of French, because it tells us about conjugation ("je" goes with "comprends," not "comprend") and negation ("ne" goes with "pas"); we wouldn't get that from regular bigrams alone.

24.1.4 Smoothing n-gram models

High-frequency n-grams like "of the" have high counts in the training corpus, so their probability estimate is likely to be accurate: with a different training corpus we would get a similar estimate. Low-frequency n-grams have low counts that are subject to random noise—they have high variance. Our models will perform better if we can smooth out that variance.

Out-of-vocabulary

Furthermore, there is always a chance that we will be asked to evaluate a text containing an unknown or **out-of-vocabulary** word: one that never appeared in the training corpus. But it would be a mistake to assign such a word a probability of zero, because then the probability of the whole sentence, $P(w_{1:N})$, would be zero.

One way to model unknown words is to modify the training corpus by replacing infrequent words with a special symbol, traditionally `<UNK>`. We could decide in advance to keep only, say, the 50,000 most common words, or all words with frequency greater than 0.0001%, and replace the others with `<UNK>`. Then compute n-gram counts for the corpus as usual, treating `<UNK>` just like any other word. When an unknown word appears in a test set, we look up its probability under `<UNK>`. Sometimes different unknown-word symbols are used for different types. For example, a string of digits might be replaced with `<NUM>`, or an email address with `<EMAIL>`. (We note that it is also advisable to have a special symbol, such as `<S>`, to mark the start (and stop) of a text. That way, when the formula for bigram probabilities asks for the word before the first word, the answer is `<S>`, not an error.)

Smoothing

Even after we've handled unknown words, we have the problem of unseen n-grams. For example, a test text might contain the phrase "colorless aquamarine ideas," three words that we may have seen individually in the training corpus, but never in that exact order. The problem is that some low-probability n-grams appear in the training corpus, while other equally low-probability n-grams happen to not appear at all. We don't want some of them to have a zero probability while others have a small positive probability; we want to apply **smoothing** to all the similar n-grams—reserving some of the probability mass of the model for never-seen n-grams, to reduce the variance of the model.

The simplest type of smoothing was suggested by Pierre-Simon Laplace in the 18th century to estimate the probability of rare events, such as the sun failing to rise tomorrow. Laplace's (incorrect) theory of the solar system suggested it was about $N = 2$ million days old. Going by the data, there were zero out of two million days when the sun failed to rise, yet we don't want to say that the probability is exactly zero. Laplace showed that if we adopt a uniform prior, and combine that with the evidence so far, we get a best estimate of $1/(N+2)$ for the probability of the sun's failure to rise tomorrow—either it will or it won't (that's the 2 in the denominator) and a uniform prior says it is as likely as not (that's the 1 in the numerator). Laplace smoothing (also called add-one smoothing) is a step in the right direction, but for many natural language applications it performs poorly.

Backoff model

Linear interpolation smoothing

Another choice is a **backoff model**, in which we start by estimating n-gram counts, but for any particular sequence that has a low (or zero) count, we back off to $(n-1)$-grams. **Linear interpolation smoothing** is a backoff model that combines trigram, bigram, and unigram models by linear interpolation. It defines the probability estimate as

$$\hat{P}(c_i|c_{i-2:i-1}) = \lambda_3 P(c_i|c_{i-2:i-1}) + \lambda_2 P(c_i|c_{i-1}) + \lambda_1 P(c_i)\,,$$

where $\lambda_3 + \lambda_2 + \lambda_1 = 1$. The parameter values λ_i can be fixed, or they can be trained with an expectation–maximization algorithm. It is also possible to have the values of λ_i depend on the counts: if we have a high count of trigrams, then we weigh them relatively more; if only a low count, then we put more weight on the bigram and unigram models.

One camp of researchers has developed ever more sophisticated smoothing techniques (such as Witten-Bell and Kneser-Ney), while another camp suggests gathering a larger corpus so that even simple smoothing techniques work well (one such approach is called "stupid backoff"). Both are getting at the same goal: reducing the variance in the language model.

24.1.5 Word representations

N-grams can give us a model that accurately predicts the probability of word sequences, telling us that, for example, "a black cat" is a more likely English phrase than "cat black a" because "a black cat" appears in about 0.000014% of the trigrams in a training corpus, while "cat black a" does not appear at all. Everything that the *n*-gram word model knows, it learned from counts of specific word sequences.

But a native speaker of English would tell a different story: "a black cat" is valid because it follows a familiar pattern (article-adjective-noun), while "cat black a" does not.

Now consider the phrase "the fulvous kitten." An English speaker could recognize this as also following the article-adjective-noun pattern (even a speaker who does not know that "fulvous" means "brownish yellow" could recognize that almost all words that end in "-ous" are adjectives). Furthermore, the speaker would recognize the close syntactic connection between "a" and "the," as well as the close semantic relation between "cat" and "kitten." Thus, the appearance of "a black cat" in the data is evidence, through generalization, that "the fulvous kitten" is also valid English.

The *n*-gram model misses this generalization because it is an *atomic* model: each word is an atom, distinct from every other word, with no internal structure. We have seen throughout this book that *factored* or *structured* models allow for more expressive power and better generalization. We will see in Section 25.1 that a factored model called **word embeddings** gives a better ability to generalize.

One type of structured word model is a **dictionary**, usually constructed through manual labor. For example, **WordNet** is an open-source, hand-curated dictionary in machine-readable format that has proven useful for many natural language applications[1] Below is the WordNet entry for "kitten:"

Dictionary

WordNet

```
"kitten" <noun.animal> ("young domestic cat") IS A: young_mammal

"kitten" <verb.body> ("give birth to kittens")
  EXAMPLE: "our cat kittened again this year"
```

WordNet will help you separate the nouns from the verbs, and get the basic categories (a kitten is a young mammal, which is a mammal, which is an animal), but it won't tell you the details of what a kitten looks like or acts like. WordNet will tell you that two subclasses of cat are *Siamese cat* and *Manx cat*, but won't tell you any more about the breeds.

[1] And even computer vision applications: WordNet provides the set of categories used by ImageNet.

Tag	Word	Description	Tag	Word	Description
CC	*and*	Coordinating conjunction	PRP$	*your*	Possessive pronoun
CD	*three*	Cardinal number	RB	*quickly*	Adverb
DT	*the*	Determiner	RBR	*quicker*	Adverb, comparative
EX	*there*	Existential there	RBS	*quickest*	Adverb, superlative
FW	*per se*	Foreign word	RP	*off*	Particle
IN	*of*	Preposition	SYM	+	Symbol
JJ	*purple*	Adjective	TO	*to*	to
JJR	*better*	Adjective, comparative	UH	*eureka*	Interjection
JJS	*best*	Adjective, superlative	VB	*talk*	Verb, base form
LS	*1*	List item marker	VBD	*talked*	Verb, past tense
MD	*should*	Modal	VBG	*talking*	Verb, gerund
NN	*kitten*	Noun, singular or mass	VBN	*talked*	Verb, past participle
NNS	*kittens*	Noun, plural	VBP	*talk*	Verb, non-3rd-sing
NNP	*Ali*	Proper noun, singular	VBZ	*talks*	Verb, 3rd-sing
NNPS	*Fords*	Proper noun, plural	WDT	*which*	Wh-determiner
PDT	*all*	Predeterminer	WP	*who*	Wh-pronoun
POS	*’s*	Possessive ending	WP$	*whose*	Possessive wh-pronoun
PRP	*you*	Personal pronoun	WRB	*where*	Wh-adverb
$	$	Dollar sign	#	#	Pound sign
“	‘	Left quote	”	’	Right quote
(	[	Left parenthesis	)	]	Right parenthesis
,	,	Comma	.	!	Sentence end
:	;	Mid-sentence punctuation			

Figure 24.1 Part-of-speech tags (with an example word for each tag) for the Penn Treebank corpus (Marcus *et al.*, 1993). Here “3rd-sing” is an abbreviation for “third person singular present tense.”

24.1.6 Part-of-speech (POS) tagging

Part of speech (POS)

One basic way to categorize words is by their **part of speech (POS)**, also called **lexical category** or **tag**: *noun, verb, adjective*, and so on. Parts of speech allow language models to capture generalizations such as “adjectives generally come before nouns in English.” (In other languages, such as French, it is the other way around (generally)).

Penn Treebank

Everyone agrees that “noun” and “verb” are parts of speech, but when we get into the details there is no one definitive list. Figure 24.1 shows the 45 tags used in the **Penn Treebank**, a corpus of over three million words of text annotated with part-of-speech tags. As we will see later, the Penn Treebank also annotates many sentences with syntactic parse trees, from which the corpus gets its name. Here is an excerpt saying that “from” is tagged as a preposition (IN), “the” as a determiner (DT), and so on:

From	the	start	,	it	took	a	person	with	great	qualities	to	succeed
IN	DT	NN	,	PRP	VBD	DT	NN	IN	JJ	NNS	TO	VB

The task of assigning a part of speech to each word in a sentence is called **part-of-speech**

tagging. Although not very interesting in its own right, it is a useful first step in many other NLP tasks, such as question answering or translation. Even for a simple task like text-to-speech synthesis, it is important to know that the noun "record" is pronounced differently from the verb "record." In this section we will see how two familiar models can be applied to the tagging task, and in Chapter 25 we will consider a third model.

Part-of-speech tagging

One common model for POS tagging is the **hidden Markov model (HMM)**. Recall from Section 14.3 that a hidden Markov model takes in a temporal sequence of evidence observations and predicts the most likely hidden states that could have produced that sequence. In the HMM example on page 491, the evidence consisted of observations of a person carrying an umbrella (or not), and the hidden state was rain (or not) in the outside world. For POS tagging, the evidence is the sequence of words, $W_{1:N}$, and the hidden states are the lexical categories, $C_{1:N}$.

The HMM is a generative model that says that the way to produce language is to start in one state, such as IN, the state for prepositions, and then make two choices: what word (such as *from*) should be emitted, and what state (such as DT) should come next. The model does not consider any context other than the current part-of-speech state, nor does it have any idea of what the sentence is actually trying to convey. And yet it is a useful model—if we apply the **Viterbi algorithm** (Section 14.2.3) to find the most probable sequence of hidden states (tags), we find that the tagging achieves very high accuracy; usually around 97%.

Viterbi algorithm

To create a HMM for POS tagging, we need the transition model, which gives the probability of one part of speech following another, $\mathbf{P}(C_t \mid C_{t-1})$, and the sensor model, $\mathbf{P}(W_t \mid C_t)$. For example, $\mathbf{P}(C_t = VB \mid C_{t-1} = MD) = 0.8$ means that given a modal verb (such as *would*), we can expect the following word to be a verb (such as *think*) with probability 0.8. Where does the 0.8 number come from? Just as with *n*-gram models, from counts in the corpus, with appropriate smoothing. It turns out that there are 13124 instances of *MD* in the Penn Treebank, and 10471 of them are followed by a *VB*.

For the sensor model, $P(W_t = would \mid C_t = MD) = 0.1$ means that when we are choosing a modal verb, we will choose *would* 10% of the time. These numbers also come from corpus counts, with smoothing.

A weakness of HMM models is that everything we know about language has to be expressed in terms of the transition and sensor models. The part of speech for the current word is determined solely by the probabilities in these two models and by the part of speech of the previous word. There is no easy way for a system developer to say, for example, that *any* word that ends in "ous" is likely an adjective, nor that in the phrase "attorney general," *attorney* is a noun, not an adjective.

Fortunately, **logistic regression** does have the ability to represent information like this. Recall from Section 19.6.5 that in a logistic regression model, the input is a vector, $\mathbf{x}$, of feature values. We then take the dot product, $\mathbf{w} \cdot \mathbf{x}$, of those features with a pretrained vector of weights $\mathbf{w}$, and transform that sum into a number between 0 and 1 that can be interpreted as the probability that the input is a positive example of a category.

The weights in the logistic regression model correspond to how predictive each feature is for each category; the weight values are learned by gradient descent. For POS tagging we would build 45 different logistic regression models, one for each part of speech, and ask each model how probable it is that the example word is a member of that category, given the feature values for that word in its particular context.

The question then is what should the features be? POS taggers typically use binary-valued features that encode information about the word being tagged, w_i (and perhaps other nearby words), as well as the category that was assigned to the previous word, c_{i-1} (and perhaps the category of earlier words). Features can depend on the exact identity of a word, some aspects of the way it is spelled, or some attribute from a dictionary entry. A set of POS tagging features might include:

w_{i-1} = "I"	w_{i+1} = "for"
w_{i-1} = "you"	c_{i-1} = IN
w_i ends with "ous"	w_i contains a hyphen
w_i ends with "ly"	w_i contains a digit
w_i starts with "un"	w_i is all uppercase
w_{i-2} = "to" and c_{i-1} = VB	w_{i-2} has attribute PRESENT
w_{i-1} = "I" and w_{i+1} = "to"	w_{i-2} has attribute PAST

For example, the word "walk" can be a noun or a verb, but in "I walk to school," the feature in the last row, left column could be used to classify "walk" as a verb (VBP). As another example, the word "cut" can be either a noun (NN), past tense verb (VBD), or present tense verb (VBP). Given the sentence "Yesterday I cut the rope," the feature in the last row, right column could help tag "cut" as VBD, while in the sentence "Now I cut the rope," the feature above that one could help tag "cut" as VBP.

All together, there might be a million features, but for any given word, only a few dozen will be nonzero. The features are usually hand-crafted by a human system designer who thinks up interesting feature templates.

Logistic regression does not have the notion of a sequence of inputs—you give it a single feature vector (information about a single word) and it produces an output (a tag). But we can force logistic regression to handle a sequence with a **greedy search**: start by choosing the most likely category for the first word, and proceed to the rest of the words in left-to-right order. At each step the category c_i is assigned according to

$$c_i = \underset{c' \in Categories}{\operatorname{argmax}} \; P(c' \mid w_{1:N}, c_{1:i-1}) .$$

That is, the classifier is allowed to look at any of the non-category features for any of the words anywhere in the sentence (because these features are all fixed), as well as any previously assigned categories.

Note that the greedy search makes a definitive category choice for each word, and then moves on to the next word; if that choice is contradicted by evidence later in the sentence, there is no possibility to go back and reverse the choice. That makes the algorithm fast. The Viterbi algorithm, in contrast, keeps a table of all possible category choices at each step, and always has the option of changing. That makes the algorithm more accurate, but slower. For both algorithms, a compromise is a **beam search**, in which we consider every possible category at each time step, but then only keep the b most likely tags, dropping the other less-likely tags. Changing b trades off speed versus accuracy.

Naive Bayes and Hidden Markov models are **generative models** (see Section 21.2.3). That is, they learn a joint probability distribution, $\mathbf{P}(W,C)$, and we can generate a random sentence by sampling from that probability distribution to get a first word (with category) of the sentence, and then adding words one at a time.

Logistic regression on the other hand is a **discriminative model**. It learns a conditional probability distribution $\mathbf{P}(C \mid W)$, meaning that it can assign categories given a sequence of words, but it can't generate random sentences. Generally, researchers have found that discriminative models have a lower error rate, perhaps because they model the intended output directly, and perhaps because they make it easier for an analyst to create additional features. However, generative models tend to converge more quickly, and so may be preferred when the available training time is short, or when there is limited training data.

24.1.7 Comparing language models

To get a feeling for what different n-gram models are like, we built unigram (i.e., bag-of-words), bigram, trigram, and 4-gram models over the words in this book and then randomly sampled word sequences from each of the four models:

- $n = 1$: *logical are as are confusion a may right tries agent goal the was*
- $n = 2$: *systems are very similar computational approach would be represented*
- $n = 3$: *planning and scheduling are integrated the success of naive Bayes model is*
- $n = 4$: *taking advantage of the structure of Bayesian networks and developed various languages for writing "templates" with logical variables, from which large networks could be constructed automatically for each problem instance*

From this small sample it should be clear that the unigram model is a very poor approximation of either English in general or an AI textbook in particular, and that the 4-gram model is imperfect but much better. Next, to demonstrate how samples switch between training sources (and definitely not just for fun) we added the text of the King James Bible to the 4-gram model yielding these random samples:

- *Prove that any 3-SAT problem can be reduced to simpler ones using the laws of thy God.*
- *Masters, give unto your servants that which is true iff both P and Q in any model m by a simple experiment: put your hand unto, ye and your households for it is pleasant.*
- *Many will intreat the LORD your God, Saying, No; but we will ignore this issue for now; Chapters 7 and 8 suggest methods for compactly representing very large belief states.*
- *And it came to pass, as if it had no successors.*
- *The direct utility estimation is just an instance of the general or algorithm in which new function symbols are constructed "on the fly." For example, the first child of the Holy Ghost.*

There is a limit to n-gram models—as n increases, they will produce language that is more fluent, but they tend to reproduce long passages from their training data verbatim, rather than generate novel text. Language models with more complex representations of words and context can do better. The rest of this chapter demonstrates how **grammar** can improve a language model, and Chapter 25 shows how deep learning methods have recently produced impressive language models. One such deep learning model, GPT-2, can produce fluent English samples when given a prompt. We gave GPT-2 the first two sentences of this paragraph as a prompt; it produced the following two samples:

- *Numerous studies have shown that the best language models that can generate language are those that use neural networks, and in particular that use deep learning to model speech. I have used one of these models to generate hundreds of sentences in multiple languages, and it has the ability to generate almost perfect translations of my training sentences. It's a great model to have, and will save you a lot of time in the future when you need to translate between many languages.*

- *These models can also do better if their samples are chosen at random, rather than by an expert in the domain. However, as a general rule, n-gram and language models that include a large number of features tend to produce text better, since these features contribute the most to language learning. To measure the performance of a model, I've made some sample sentences that I'd like to train to better understand speech and language models. You can read them to get a grasp of how the model predicts.*

We see that these passages are diverse and grammatically fluent; moreover, they stick to topics that are relevant to the prompt sentences. But the sentences do not build on each other to advance a coherent thesis. The GPT-2 language model is known as a **transformer model**, which will be covered in Section 25.4; further examples from GPT-2 are in Figure 25.14. Another transformer model is the Conditional Transformer Language, CTRL. It can be controlled more flexibly; in the following samples CTRL was asked to generate text in the category *product reviews*, with a rating of 1 and of 4 (out of 5): specified rating (out of 5):

- **1.0**: *I bought this for my son who is a huge fan of the show. He was so excited to get it and when he opened it, we were all very disappointed. The quality of the product is terrible. It looks like something you would buy at a dollar store.*
- **4.0**: *I bought this for my husband and he loves it. He has a small wrist so it is hard to find watches that fit him well. This one fits perfectly.*

24.2 Grammar

In Chapter 7 we used Backus–Naur Form (BNF) to write down a grammar for the language of first-order logic. A **grammar** is a set of rules that defines the tree structure of allowable phrases, and a **language** is the set of sentences that follow those rules.

Natural languages do not work exactly like the formal language of first-order logic—they do not have a hard boundary between allowable and unallowable sentences, nor do they have a single definitive tree structure for each sentence. However, hierarchical structure *is* important in natural language. The word "Stocks" in "Stocks rallied on Monday" is not just a word, nor is it just a *noun*; in this sentence it also comprises a *noun phrase*, which is the subject of the following *verb phrase*. **Syntactic categories** such as *noun phrase* or *verb phrase* help to constrain the probable words at each point within a sentence, and the **phrase structure** provides a framework for the meaning or **semantics** of the sentence.

Syntactic category

Phrase structure

There are many competing language models based on the idea of hierarchical syntactic structure; in this section we will describe a popular model called the **probabilistic context-free grammar**, or PCFG. A probabilistic grammar assigns a probability to each string, and "context-free" means that any rule can be used in any context: the rules for a noun phrase at the beginning of a sentence are the same as for another noun phrase later in the sentence, and if the same phrase occurs in two locations, it must have the same probability each time. We will define a PCFG grammar for a tiny fragment of English that is suitable for communication between agents exploring the wumpus world. We call this language $\mathcal{E}_0$ (see Figure 24.2). A grammar rule such as

Probabilistic context-free grammar

$$\begin{array}{lcll} Adjs & \rightarrow & Adjective & [0.80] \\ & | & Adjective\ Adjs & [0.20] \end{array}$$

means that the syntactic category *Adjs* can consist of either a single *Adjective*, with probability 0.80, or of an *Adjective* followed by a string that constitutes an *Adjs*, with probability 0.20.

S	→	*NP VP*	[0.90]	I + feel a breeze
	|	*S Conj S*	[0.10]	I feel a breeze + and + It stinks
NP	→	*Pronoun*	[0.25]	I
	|	*Name*	[0.10]	Ali
	|	*Noun*	[0.10]	pits
	|	*Article Noun*	[0.25]	the + wumpus
	|	*Article Adjs Noun*	[0.05]	the + smelly dead + wumpus
	|	*Digit Digit*	[0.05]	3 4
	|	*NP PP*	[0.10]	the wumpus + in 1 3
	|	*NP RelClause*	[0.05]	the wumpus + that is smelly
	|	*NP Conj NP*	[0.05]	the wumpus + and + I
VP	→	*Verb*	[0.40]	stinks
	|	*VP NP*	[0.35]	feel + a breeze
	|	*VP Adjective*	[0.05]	smells + dead
	|	*VP PP*	[0.10]	is + in 1 3
	|	*VP Adverb*	[0.10]	go + ahead
Adjs	→	*Adjective*	[0.80]	smelly
	|	*Adjective Adjs*	[0.20]	smelly + dead
PP	→	*Prep NP*	[1.00]	to + the east
RelClause	→	*RelPro VP*	[1.00]	that + is smelly

Figure 24.2 The grammar for $\mathcal{E}_0$, with example phrases for each rule. The syntactic categories are sentence (*S*), noun phrase (*NP*), verb phrase (*VP*), list of adjectives (*Adjs*), prepositional phrase (*PP*), and relative clause (*RelClause*).

Noun	→	**stench** [0.05] | **breeze** [0.10] | **wumpus** [0.15] | **pits** [0.05] | . . .
Verb	→	**is** [0.10] | **feel** [0.10] | **smells** [0.10] | **stinks** [0.05] | . . .
Adjective	→	**right** [0.10] | **dead** [0.05] | **smelly** [0.02] | **breezy** [0.02] . . .
Adverb	→	**here** [0.05] | **ahead** [0.05] | **nearby** [0.02] | . . .
Pronoun	→	**me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] | . . .
RelPro	→	**that** [0.40] | **which** [0.15] | **who** [0.20] | **whom** [0.02] | . . .
Name	→	**Ali** [0.01] | **Bo** [0.01] | **Boston** [0.01] | . . .
Article	→	**the** [0.40] | **a** [0.30] | **an** [0.10] | **every** [0.05] | . . .
Prep	→	**to** [0.20] | **in** [0.10] | **on** [0.05] | **near** [0.10] | . . .
Conj	→	**and** [0.50] | **or** [0.10] | **but** [0.20] | **yet** [0.02] | . . .
Digit	→	**0** [0.20] | **1** [0.20] | **2** [0.20] | **3** [0.20] | **4** [0.20] | . . .

Figure 24.3 The lexicon for $\mathcal{E}_0$. *RelPro* is short for relative pronoun, *Prep* for preposition, and *Conj* for conjunction. The sum of the probabilities for each category is 1.

Overgeneration
Undergeneration

Unfortunately, the grammar **overgenerates**: that is, it generates sentences that are not grammatical, such as “Me go I.” It also **undergenerates**: there are many sentences of English that it rejects, such as “I think the wumpus is smelly.” We will see how to learn a better grammar later; for now we concentrate on what we can do with this very simple grammar.

24.2.1 The lexicon of $\mathscr{E}_0$

Lexicon

The **lexicon**, or list of allowable words, is defined in Figure 24.3. Each of the lexical categories ends in ... to indicate that there are other words in the category. For nouns, names, verbs, adjectives, and adverbs, it is infeasible even in principle to list all the words. Not only are there tens of thousands of members in each class, but new ones—like *humblebrag* or *microbiome*—are being added constantly. These five categories are called **open classes**. Pronouns, relative pronouns, articles, prepositions, and conjunctions are called **closed classes**; they have a small number of words (a dozen or so), and change over the course of centuries, not months. For example, “thee” and “thou” were commonly used pronouns in the 17th century, were on the decline in the 19th century, and are seen today only in poetry and some regional dialects.

Open class
Closed class

24.3 Parsing

Parsing

Parsing is the process of analyzing a string of words to uncover its phrase structure, according to the rules of a grammar. We can think of it as a **search** for a valid parse tree whose leaves are the words of the string. Figure 24.4 shows that we can start with the S symbol and search top down, or we can start with the words and search bottom up. Pure top-down or bottom-up parsing strategies can be inefficient, however, because they can end up repeating effort in areas of the search space that lead to dead ends. Consider the following two sentences:

Have the students in section 2 of Computer Science 101 take the exam.
Have the students in section 2 of Computer Science 101 taken the exam?

Even though they share the first 10 words, these sentences have very different parses, because the first is a command and the second is a question. A left-to-right parsing algorithm would have to guess whether the first word is part of a command or a question and will not be able to tell if the guess is correct until at least the eleventh word, *take* or *taken*. If the algorithm guesses wrong, it will have to backtrack all the way to the first word and reanalyze the whole sentence under the other interpretation.

To avoid this source of inefficiency we can use **dynamic programming**: every time we analyze a substring, store the results so we won’t have to reanalyze it later. For example, once we discover that “the students in section 2 of Computer Science 101” is an NP, we can record that result in a data structure known as a **chart**. An algorithm that does this is called a **chart parser**. Because we are dealing with context-free grammars, any phrase that was found in the context of one branch of the search tree can work just as well in any other branch of the search tree. There are many types of chart parsers; we describe a probabilistic version of a bottom-up chart parsing algorithm called the **CYK algorithm**, after its inventors, Ali Cocke, Daniel Younger, and Tadeo Kasami.[2]

Chart parser

CYK algorithm

[2] Sometimes the authors are credited in the order CKY.

List of items	Rule
S	
NP VP	*S* → *NP VP*
NP VP Adjective	*VP* → *VP Adjective*
NP Verb Adjective	*VP* → *Verb*
NP Verb **dead**	*Adjective* → **dead**
NP **is dead**	*Verb* → **is**
Article Noun **is dead**	*NP* → *Article Noun*
Article **wumpus is dead**	*Noun* → **wumpus**
the wumpus is dead	*Article* → **the**

Figure 24.4 Parsing the string "The wumpus is dead" as a sentence, according to the grammar $\mathcal{E}_0$. Viewed as a top-down parse, we start with *S*, and on each step match one nonterminal *X* with a rule of the form $(X \to Y \ldots)$ and replace *X* in the list of items with $Y \ldots$; for example replacing *S* with the sequence *NP VP*. Viewed as a bottom-up parse, we start with the words "the wumpus is dead", and on each step match a string of tokens such as $(Y \ldots)$ against a rule of the form $(X \to Y \ldots)$ and replace the tokens with *X*; for example replacing "the" with *Article* or *Article Noun* with *NP*.

The CYK algorithm is shown in Figure 24.5. It requires a grammar with all rules in one of two very specific formats: lexical rules of the form $X \to$ **word** $[p]$, and syntactic rules of the form $X \to Y\ Z\ [p]$, with exactly two categories on the right-hand side. This grammar format, called **Chomsky Normal Form**, may seem restrictive, but it is not: any context-free grammar can be automatically transformed into Chomsky Normal Form. Exercise 24.CNFX leads you through the process.

Chomsky Normal Form

The CYK algorithm uses space of $O(n^2m)$ for the *P* and *T* tables, where *n* is the number of words in the sentence, and *m* is the number of nonterminal symbols in the grammar, and takes time $O(n^3m)$. If we want an algorithm that is guaranteed to work for all possible context-free grammars, then we can't do any better than that. But actually we only want to parse natural languages, not all possible grammars. Natural languages have evolved to be easy to understand in real time, not to be as tricky as possible, so it seems that they should be amenable to a faster parsing algorithm.

To try to get to $O(n)$, we can apply A^* search in a fairly straightforward way: each state is a list of items (words or categories), as shown in Figure 24.4. The start state is a list of words, and a goal state is the single item *S*. The cost of a state is the inverse of its probability as defined by the rules applied so far, and there are various heuristics to estimate the remaining distance to the goal; the best heuristics in current use come from machine learning applied to a corpus of sentences.

With the A^* algorithm we don't have to search the entire state space, and we are guaranteed that the first parse found will be the most probable (assuming an admissible heuristic). This will usually be faster than CYK, but (depending on the details of the grammar) still slower than $O(n)$. An example result of a parse is shown in Figure 24.6.

Just as with part-of-speech tagging, we can use a **beam search** for parsing, where at any time we consider only the *b* most probable alternative parses. This means we are not

```
function CYK-PARSE($words$, $grammar$) returns a table of parse trees
  inputs: $words$, a list of words
          $grammar$, a structure with LEXICALRULES and GRAMMARRULES
  $T \leftarrow$ a table      // $T[X, i, k]$ is most probable $X$ tree spanning $words_{i:k}$
  $P \leftarrow$ a table, initially all 0      // $P[X, i, k]$ is probability of tree $T[X, i, k]$
  // Insert lexical categories for each word.
  for $i$ = 1 to LEN($words$) do
    for each $(X, p)$ in $grammar$.LEXICALRULES($words_i$) do
      $P[X, i, i] \leftarrow$ p
      $T[X, i, i] \leftarrow$ TREE($X$, $words_i$)
  // Construct $X_{i:k}$ from $Y_{i:j} + Z_{j+1:k}$, shortest spans first.
  for each $(i, j, k)$ in SUBSPANS(LEN($words$)) do
    for each $(X, Y, Z, p)$ in $grammar$.GRAMMARRULES do
      $PYZ \leftarrow P[Y, i, j] \times P[Z, j+1, k] \times p$
      if $PYZ > P[X, i, k]$ do
        $P[X, i, k] \leftarrow PYZ$
        $T[X, i, k] \leftarrow$ TREE($X$, $T[Y, i, j]$, $T[Z, j+1, k]$)
  return $T$

function SUBSPANS(N) yields $(i, j, k)$ tuples
  for $length$ = 2 to $N$ do
    for $i$ = 1 to $N + 1 - length$ do
      $k \leftarrow i + length - 1$
      for $j = i$ to $k - 1$ do
        yield $(i, j, k)$
```

Figure 24.5 The CYK algorithm for parsing. Given a sequence of words, it finds the most probable parse tree for the sequence and its subsequences. The table $P[X,i,k]$ gives the probability of the most probable tree of category X spanning $words_{i:k}$. The output table $T[X, i, k]$ contains the most probable tree of category X spanning positions i to k inclusive. The function SUBSPANS returns all tuples (i,j,k) covering a span of $words_{i:k}$, with $i \leq j < k$, listing the tuples by increasing length of the $i:k$ span, so that when we go to combine two shorter spans into a longer one, the shorter spans are already in the table. LEXICALRULES($word$) returns a collection of (X,p) pairs, one for each rule of the form $X \rightarrow word$ [p], and GRAMMARRULES gives (X,Y,Z,p) tuples, one for each grammar rule of the form $X \rightarrow Y\ Z\ [p]$.

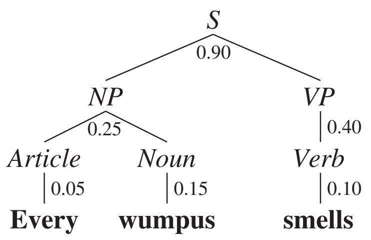


Figure 24.6 Parse tree for the sentence "Every wumpus smells" according to the grammar $\mathcal{E}_0$. Each interior node of the tree is labeled with its probability. The probability of the tree as a whole is $0.9 \times 0.25 \times 0.05 \times 0.15 \times 0.40 \times 0.10 = 0.0000675$. The tree can also be written in linear form as [S [NP [$Article$ **every**] [$Noun$ **wumpus**]][VP [$Verb$ **smells**]]].

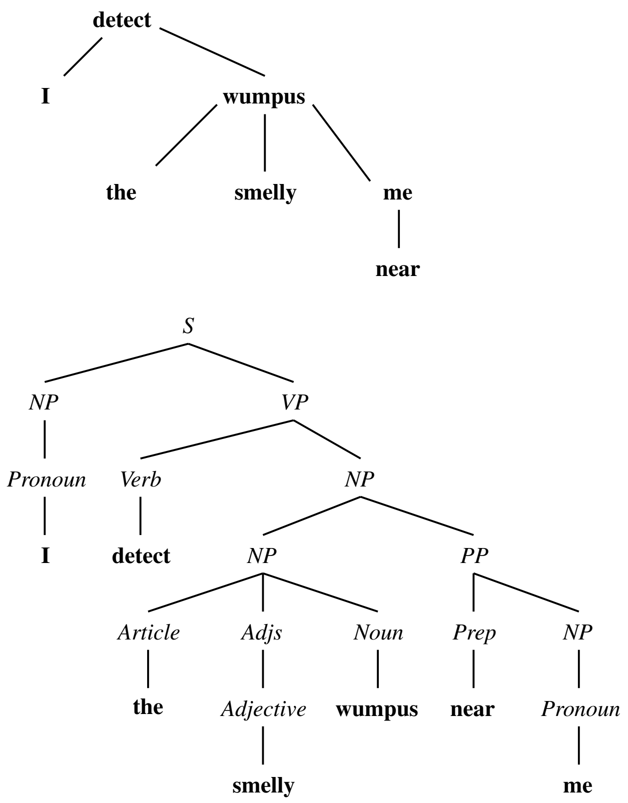


Figure 24.7 A dependency-style parse (top) and the corresponding phrase structure parse (bottom) for the sentence *I detect the smelly wumpus near me.*

guaranteed to find the parse with highest probability, but (with a careful implementation) the parser can operate in $O(n)$ time and still finds the best parse most of the time.

Deterministic parser

Shift-reduce parsing

A beam search parser with $b = 1$ is called a **deterministic parser**. One popular deterministic approach is **shift-reduce parsing**, in which we go through the sentence word by word, choosing at each point whether to shift the word onto a stack of constituents, or to reduce the top constituent(s) on the stack according to a grammar rule. Each style of parsing has its adherents within the NLP community. Even though it is possible to transform a shift-reduce system into a PCFG (and vice versa), when you apply machine learning to the problem of inducing a grammar, the inductive bias and hence the generalizations that each system will make will be different (Abney *et al.*, 1999).

24.3.1 Dependency parsing

Dependency grammar

There is a widely used alternative syntactic approach called **dependency grammar**, which assumes that syntactic structure is formed by binary relations between lexical items, without a need for syntactic constituents. Figure 24.7 shows a sentence with a dependency parse and a phrase structure parse.

In one sense, dependency grammar and phrase structure grammar are just notational variants. If the phrase structure tree is annotated with the head of each phrase, you can recover

```
[ [S [NP-2 Her eyes]
     [VP were
         [VP glazed
             [NP *-2]
             [SBAR-ADV as if
                   [S [NP she]
                      [VP did n't
                          [VP [VP hear [NP *-1]]
                              or
                              [VP [ADVP even] see [NP *-1]]
                              [NP-1 him]]]]]]]]
  .]
```

Figure 24.8 Annotated tree for the sentence "Her eyes were glazed as if she didn't hear or even see him." from the Penn Treebank. Note a grammatical phenomenon we have not covered yet: the movement of a phrase from one part of the tree to another. This tree analyzes the phrase "hear or even see him" as consisting of two constituent *VP*s, [*VP* **hear** [*NP* *-1]] and [*VP* [*ADVP* **even**] **see** [*NP* *-1]], both of which have a missing object, denoted *-1, which refers to the *NP* labeled elsewhere in the tree as [*NP*-1 **him**]. Similarly, the [*NP* *-2] refers to the [*NP*-2 **Her eyes**].

the dependency tree from it. In the other direction, we can convert a dependency tree into a phrase structure tree by introducing arbitrary categories (although we might not always get a natural-looking tree this way).

Therefore we wouldn't prefer one notation over the other because one is more powerful; rather we would prefer one because it is more natural—either more familiar for the human developers of a system, or more natural for a machine learning system which will have to learn the structures. In general, phrase structure trees are natural for languages (like English) with mostly fixed word order; dependency trees are natural for languages (such as Latin) with mostly free word order, where the order of words is determined more by pragmatics than by syntactic categories.

The popularity of dependency grammar today stems in large part from the Universal Dependencies project (Nivre *et al.*, 2016), an open-source treebank project that defines a set of relations and provides millions of parsed sentences in over 70 languages.

24.3.2 Learning a parser from examples

Building a grammar for a significant portion of English is laborious and error prone. This suggests that it would be better to **learn** the grammar rules (and probabilities) rather than writing them down by hand. To apply supervised learning, we need input/output pairs of sentences and their parse trees. The Penn Treebank is the best known source of such data, with over 100 thousand sentences annotated with parse-tree structure. Figure 24.8 shows an annotated tree from the Penn Treebank.

Given a treebank, we can create a PCFG just by counting the number of times each node-type appears in a tree (with the usual caveats about smoothing low counts). In Figure 24.8, there are two nodes of the form $[S[NP\ldots][VP\ldots]]$. We would count these, and all the other subtrees with root S in the corpus. If there are 1000 S nodes of which 600 are of this form, then we create the rule:

$$S \rightarrow NP\ VP\ [0.6]\,.$$

All together, the Penn Treebank has over 10,000 different node types. This reflects the fact that English is a complex language, but it also indicates that the annotators who created the treebank favored flat trees, perhaps flatter than we would like. For example, the phrase "the good and the bad" is parsed as a single noun phrase rather than as two conjoined noun phrases, giving us the rule:

$$NP \rightarrow Article\ Noun\ Conjunction\ Article\ Noun\ \ .$$

There are hundreds of similar rules that define a noun phrase as a string of categories with a conjunction somewhere in the middle; a more concise grammar could capture all the conjoined noun phrase rules with the single rule

$$NP \rightarrow NP\ Conjunction\ NP\,.$$

Bod *et al.* (2003) and Bod (2008) show how to automatically recover generalized rules like this, greatly reducing the number of rules that come out of the treebank, and creating a grammar that ends up generalizing better for previously unseen sentences. They call their approach **data-oriented parsing**.

We have seen that treebanks are not perfect—they contain errors, and have idiosyncratic parses. It is also clear that creating a treebank requires a lot of hard work; that means that treebanks will remain relatively small in size, compared to all the text that has not been annotated with trees. An alternative approach is **unsupervised parsing**, in which we learn a new grammar (or improve an existing grammar) using a corpus of sentences without trees.

Unsupervised parsing

The **inside–outside algorithm** (Dodd, 1988), which we will not cover here, learns to estimate the probabilities in a PCFG from example sentences without trees, similar to the way the forward-backward algorithm (Figure 14.4) estimates probabilities. Spitkovsky *et al.* (2010a) describe an unsupervised learning approach that uses **curriculum learning**: start with the easy part of the curriculum—short unambiguous 2-word sentences like "He left" can be easily parsed based on prior knowledge or annotations. Each new parse of a short sentence extends the system's knowledge so that it can eventually tackle 3-word, then 4-word, and eventually 40-word sentences.

Curriculum learning

We can also use **semisupervised parsing**, in which we start with a small number of trees as data to build an initial grammar, then add a large number of unparsed sentences to improve the grammar. The semisupervised approach can make use of **partial bracketing**: we can use widely available text that has been marked up by the authors, not by linguistic experts, with a partial tree-like structure, in the form of HTML or similar annotations. In HTML text most brackets correspond to a syntactic component, so partial bracketing can help learn a grammar (Pereira and Schabes, 1992; Spitkovsky *et al.*, 2010b). Consider this HTML text from a newspaper article:

Semisupervised parsing

Partial bracketing

```
  In 1998, however, as I <a>established in
<i>The New Republic</i></a> and Bill Clinton just
<a>confirmed in his memoirs</a>, Netanyahu changed his mind
```

The words surrounded by `<i></i>` tags form a noun phrase, and the two strings of words surrounded by `<a></a>` tags each form verb phrases.

24.4 Augmented Grammars

So far we have dealt with **context-free grammars**. But not every *NP* can appear in every context with equal probability. The sentence "I ate a banana" is fine, but "Me ate a banana" is ungrammatical, and "I ate a bandanna" is unlikely.

The issue is that our grammar is focused on lexical categories, like *Pronoun*, but while "I" and "me" are both pronouns, only "I" can be the subject of a sentence. Similarly, "banana" and "bandanna" are both nouns, but the former is much more likely to be object of "ate". Linguists say that the pronoun "I" is in the subjective case (i.e., is the subject of a verb) and "me" is in the objective case[3] (i.e., is the object of a verb). They also say that "I" is in the first person ("you" is second person, and "she" is third person) and is singular ("we" is plural). A category like *Pronoun* that has been augmented with features like "subjective case, first person singular" is called a **subcategory**.

Subcategory

In this section we show how a grammar can represent this kind of knowledge to make finer-grained distinctions about which sentences are more likely. We will also show how to construct a representation of the **semantics** of a phrase, in a compositional way. All of this will be accomplished with an **augmented grammar** in which the nonterminals are not just atomic symbols like *Pronoun* or *NP*, but are structured representations. For example, the noun phrase "I" could be represented as $NP(Sbj, 1S, Speaker)$, which means "a noun phrase that is in the subjective case, first person singular, and whose meaning is the speaker of the sentence." In contrast, "me" would be represented as $NP(Obj, 1S, Speaker)$, marking the fact that it is in the objective case.

Augmented grammar

Consider the sequence "*Noun* and *Noun* or *Noun*," which can be parsed either as "[*Noun* and *Noun*] or *Noun*," or as "*Noun* and [*Noun* or *Noun*]." Our context-free grammar has no way to express a preference for one parse over the other, because the rule for conjoined *NP*s, $NP \rightarrow NP\ Conjunction\ NP [0.05]$, will give the same probability to each parse. We would like a grammar that prefers the parses "[[spaghetti and meatballs] or lasagna]" and "[spaghetti and [pie or cake]]" over the alternative bracketing for each of these phrases.

Lexicalized PCFG

A **lexicalized PCFG** is a type of augmented grammar that allows us to assign probabilities based on properties of the words in a phrase other than just the syntactic categories. The data would be very sparse indeed if the probability of, say, a 40-word sentence depended on *all* 40 words—this is the same problem we noted with n-grams. To simplify, we introduce the notion of the **head** of a phrase—the most important word. Thus, "banana" is the head of the *NP* "a banana" and "ate" is the head of the *VP* "ate a banana." The notation $VP(v)$ denotes a phrase with category *VP* whose head word is v. Here is a lexicalized PCFG:

Head

$$\begin{array}{lll}
VP(v) & \rightarrow\ Verb(v)\ NP(n) & [P_1(v,n)] \\
VP(v) & \rightarrow\ Verb(v) & [P_2(v)] \\
NP(n) & \rightarrow\ Article(a)\ Adjs(j)\ Noun(n) & [P_3(n,a)] \\
NP(n) & \rightarrow\ NP(n)\ Conjunction(c)\ NP(m) & [P_4(n,c,m)] \\
Verb(\textbf{ate}) & \rightarrow\ \textbf{ate} & [0.002] \\
Noun(\textbf{banana}) & \rightarrow\ \textbf{banana} & [0.0007]
\end{array}$$

[3] The subjective case is also sometimes called the nominative case and the objective case is sometimes called the accusative case. Many languages also make another distinction with a dative case for words in the indirect object position.

$$\begin{array}{rcl}
S(v) & \rightarrow & NP(Sbj,pn,n)\ VP(pn,v) \mid \ldots \\
NP(c,pn,n) & \rightarrow & Pronoun(c,pn,n) \mid Noun(c,pn,n) \mid \ldots \\
VP(pn,v) & \rightarrow & Verb(pn,v)\ NP(Obj,pn,n) \mid \ldots \\
PP(head) & \rightarrow & Prep(head)\ NP(Obj,pn,h) \\
Pronoun(Sbj,1S,\mathbf{I}) & \rightarrow & \mathbf{I} \\
Pronoun(Sbj,1P,\mathbf{we}) & \rightarrow & \mathbf{we} \\
Pronoun(Obj,1S,\mathbf{me}) & \rightarrow & \mathbf{me} \\
Pronoun(Obj,3P,\mathbf{them}) & \rightarrow & \mathbf{them} \\
\\
Verb(3S,\mathbf{see}) & \rightarrow & \mathbf{see}
\end{array}$$

Figure 24.9 Part of an augmented grammar that handles case agreement, subject–verb agreement, and head words. Capitalized names are constants: *Sbj*, and *Obj* for subjective and objective case; *1S* for first person singular; *1P* and *3P* for first and third person plural. As usual, lowercase names are variables. For simplicity, the probabilities have been omitted.

Here $P_1(v,n)$ means the probability of a *VP* headed by v joining with an *NP* headed by n to form a *VP*. We can specify that "ate a banana" is more probable than "ate a bandanna" by ensuring that $P_1(ate, banana) > P_1(ate, bandanna)$. Note that since we are considering only phrase heads, the distinction between "ate a banana" and "ate a rancid banana" will not be caught by P_1. Conceptually, P_1 is a huge table of probabilities: if there are 5,000 verbs and 10,000 nouns in the vocabulary, then P_1 requires 50 million entries, but most of them will not be stored explicitly; rather they will be derived from smoothing and backoff. For example, we can back off from $P_1(v,n)$ to a model that depends only on v. Such a model would require 10,000 times fewer parameters, but can still capture important regularities, such as the fact that a transitive verb like "ate" is more likely to be followed by an *NP* (regardless of the head) than an intransitive verb like "sleep."

We saw in Section 24.2 that the simple grammar for $\mathcal{E}_0$ overgenerates, producing nonsentences such as "I saw she" or "I sees her." To avoid this problem, our grammar would have to know that "her," not "she," is a valid object of "saw" (or of any other verb) and that "see," not "sees," is the form of the verb that accompanies the subject "I."

We could encode these facts completely in the probability entries, for example making $P_1(v, she)$ be a very small number, for all verbs v. But it is more concise and modular to augment the category *NP* with additional variables: $NP(c,pn,n)$ is used to represent a noun phrase with case c (subjective or objective), person and number pn (e.g., third person singular), and head noun n. Figure 24.9 shows an augmented lexicalized grammar that handles these additional variables. Let's consider one grammar rule in detail:

$$S(v) \rightarrow NP(Sbj,pn,n)\ VP(pn,v)\ \ [P_5(n,v)]\,.$$

This rule says that when an *NP* is followed by a *VP* they can form an *S*, but only if the *NP* has the subjective (*Sbj*) case and the person and number (*pn*) of the *NP* and *VP* are identical. (We say that they are *in agreement*.) If that holds, then we have an *S* whose head is the verb from the *VP*. Here is an example lexical rule,

$$Pronoun(Sbj,1S,I) \rightarrow \mathbf{I}\ \ [0.005]$$

which says that "I" is a *Pronoun* in the subjective case, first-person singular, with head "I."

$$
\begin{aligned}
&Exp(op(x_1,x_2)) \rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\\
&Exp(x) \rightarrow \textbf{(}\ Exp(x)\ \textbf{)}\\
&Exp(x) \rightarrow Number(x)\\
&Number(x) \rightarrow Digit(x)\\
&Number(10 \times x_1 + x_2) \rightarrow Number(x_1)\ Digit(x_2)\\
&Operator(+) \rightarrow \textbf{+}\\
&Operator(-) \rightarrow \textbf{-}\\
&Operator(\times) \rightarrow \times\\
&Operator(\div) \rightarrow \div\\
&Digit(0) \rightarrow \textbf{0}\\
&Digit(1) \rightarrow \textbf{1}\\
&\ldots
\end{aligned}
$$

Figure 24.10 A grammar for arithmetic expressions, augmented with semantics. Each variable x_i represents the semantics of a constituent.

24.4.1 Semantic interpretation

To show how to add semantics to a grammar, we start with an example that is simpler than English: the semantics of arithmetic expressions. Figure 24.10 shows a grammar for arithmetic expressions, where each rule is augmented with a single argument indicating the semantic interpretation of the phrase. The semantics of a digit such as “3” is the digit itself. The semantics of the expression “3 + 4” is the operator “+” applied to the semantics of the phrases “3” and “4.” The grammar rules obey the principle of **compositional semantics**—the semantics of a phrase is a function of the semantics of the subphrases. Figure 24.11 shows the parse tree for $3+(4 \div 2)$ according to this grammar. The root of the parse tree is $Exp(5)$, an expression whose semantic interpretation is 5.

Compositional semantics

Now let’s move on to the semantics of English, or at least a tiny portion of it. We will use first-order logic for our semantic representation. So the simple sentence “Ali loves Bo” should get the semantic representation $Loves(Ali, Bo)$. But what about the constituent phrases? We can represent the *NP* “Ali” with the logical term *Ali*. But the *VP* “loves Bo” is neither a logical term nor a complete logical sentence. Intuitively, “loves Bo” is a description that might or might not apply to a particular person. (In this case, it applies to Ali.) This means that “loves Bo” is a **predicate** that, when combined with a term that represents a person, yields a complete logical sentence.

Using the λ-notation (see page 277), we can represent “loves Bo” as the predicate

$$\lambda x\, Loves(x, Bo)\,.$$

Now we need a rule that says “an *NP* with semantics *n* followed by a *VP* with semantics *pred* yields a sentence whose semantics is the result of applying *pred* to *n*:”

$$S(pred(n)) \rightarrow NP(n)\ VP(pred)\,.$$

The rule tells us that the semantic interpretation of “Ali loves Bo” is

$$(\lambda x\, Loves(x, Bo))(Ali)\,,$$

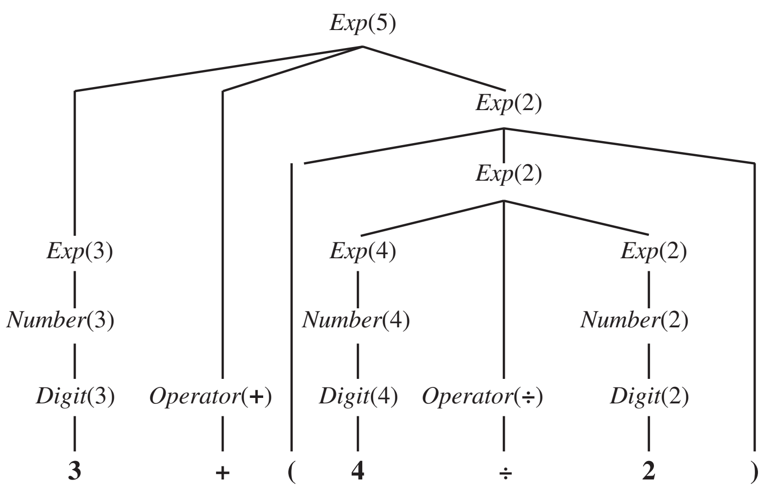
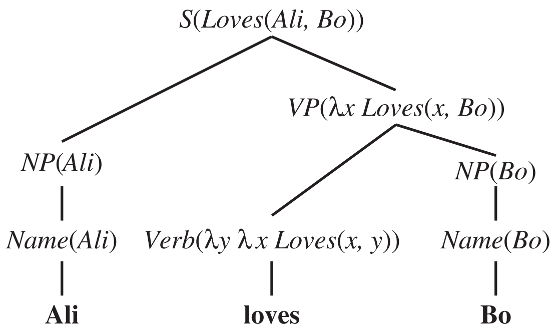


Figure 24.11 Parse tree with semantic interpretations for the string "$3+(4\div 2)$".

$$S(pred(n)) \rightarrow NP(n)\ VP(pred)$$
$$VP(pred(n)) \rightarrow Verb(pred)\ NP(n)$$
$$NP(n) \rightarrow Name(n)$$

$$Name(Ali) \rightarrow \textbf{Ali}$$
$$Name(Bo) \rightarrow \textbf{Bo}$$
$$Verb(\lambda y\ \lambda x\ Loves(x,y)) \rightarrow \textbf{loves}$$

(a)



(b)

Figure 24.12 (a) A grammar that can derive a parse tree and semantic interpretation for "Ali loves Bo" (and three other sentences). Each category is augmented with a single argument representing the semantics. (b) A parse tree with semantic interpretations for the string "Ali loves Bo."

which is equivalent to $Loves(Ali,Bo)$. Technically, we say that this is a β-reduction of the lambda function application.

The rest of the semantics follows in a straightforward way from the choices we have made so far. Because *VP*s are represented as predicates, verbs should be predicates as well. The verb "loves" is represented as $\lambda y\ \lambda x\ Loves(x,y)$, the predicate that, when given the argument *Bo*, returns the predicate $\lambda x\ Loves(x,Bo)$. We end up with the grammar and parse tree shown in Figure 24.12. In a more complete grammar, we would put all the augmentations (semantics, case, person-number, and head) together into one set of rules. Here we show only the semantic augmentation to make it clearer how the rules work.

24.4.2 Learning semantic grammars

Unfortunately, the Penn Treebank does not include semantic representations of its sentences, just syntactic trees. So if we are going to learn a semantic grammar, we will need a different source of examples. Zettlemoyer and Collins (2005) describe a system that learns a grammar for a question-answering system from examples that consist of a sentence paired with the semantic form for the sentence:

- **Sentence**: What states border Texas?
- **Logical Form**: $\lambda x.state(x) \land \lambda x.borders(x, Texas)$

Given a large collection of pairs like this and a little bit of hand-coded knowledge for each new domain, the system generates plausible lexical entries (for example, that "Texas" and "state" are nouns such that $state(Texas)$ is true), and simultaneously learns parameters for a grammar that allows the system to parse sentences into semantic representations. Zettlemoyer and Collins's system achieved 79% accuracy on two different test sets of unseen sentences. Zhao and Huang (2015) demonstrate a shift-reduce parser that runs faster, and achieves 85% to 89% accuracy.

A limitation of these systems is that the training data includes logical forms. These are expensive to create, requiring human annotators with specialized expertise—not everyone understands the subtleties of lambda calculus and predicate logic. It is much easier to gather examples of question/answer pairs:

- **Question**: What states border Texas?
- **Answer**: Louisiana, Arkansas, Oklahoma, New Mexico.

- **Question**: How many times would Rhode Island fit into California?
- **Answer**: 135

Such question/answer pairs are quite common on the Web, so a large database can be put together without human experts. Using this large source of data it is possible to build parsers that outperform those that use a small database of annotated logical forms (Liang *et al.*, 2011; Liang and Potts, 2015). The key approach described in these papers is to invent an internal logical form that is compositional but does not allow an exponentially large search space.

24.5 Complications of Real Natural Language

The grammar of real English is endlessly complex (and other languages are equally complex). We will briefly mention some of the topics that contribute to this complexity.

Quantification

Quantification: Consider the sentence "Every agent feels a breeze." The sentence has only one syntactic parse under $\mathcal{E}_0$, but it is semantically ambiguous: is there one breeze that is felt by all the agents, or does each agent feel a separate personal breeze? The two interpretations can be represented as

$$\begin{aligned}
&\forall a\ \ a \in Agents \Rightarrow \\
&\qquad \exists b\ \ b \in Breezes \land Feel(a,b)\ ; \\
&\exists b\ \ b \in Breezes \land \forall a\ \ a \in Agents \Rightarrow \\
&\qquad Feel(a,b)\,.
\end{aligned}$$

One standard approach to quantification is for the grammar to define not an actual logical semantic sentence, but rather a **quasi-logical form** that is then turned into a logical sentence by algorithms outside of the parsing process. Those algorithms can have preference rules for choosing one quantifier scope over another—preferences that need not be reflected directly in the grammar.

Quasi-logical form

Pragmatics: We have shown how an agent can perceive a string of words and use a grammar to derive a set of possible semantic interpretations. Now we address the problem of completing the interpretation by adding context-dependent information about the current situation. The most obvious need for pragmatic information is in resolving the meaning of **indexicals**, which are phrases that refer directly to the current situation. For example, in the sentence "I am in Boston today," both "I" and "today" are indexicals. The word "I" would be represented by *Speaker*, a fluent that refers to different objects at different times, and it would be up to the hearer to resolve the referent of the fluent—that is not considered part of the grammar but rather an issue of pragmatics.

Pragmatics

Indexical

Another part of pragmatics is interpreting the speaker's intent. The speaker's utterance is considered a **speech act**, and it is up to the hearer to decipher what type of action it is—a question, a statement, a promise, a warning, a command, and so on. A command such as "go to 2 2" implicitly refers to the hearer. So far, our grammar for *S* covers only declarative sentences. We can extend it to cover commands—a command is a verb phrase where the subject is implicitly the hearer of the command:

Speech act

$$S(Command(pred(Hearer))) \rightarrow VP(pred)\,.$$

Long-distance dependencies: In Figure 24.8 we saw that "she didn't hear or even see him" was parsed with two gaps where an *NP* is missing, but refers to the *NP* "him." We can use the symbol ␣ to represent the gaps: "she didn't [hear ␣ or even see ␣] him." In general, the distance between the gap and the *NP* it refers to can be arbitrarily long: in "Who did the agent tell you to give the gold to ␣?" the gap refers to "Who," which is 11 words away.

Long-distance dependencies

A complex system of augmented rules can be used to make sure that the missing *NP*s match up properly. The rules are complex; for example, you can't have a gap in one branch of an *NP* conjunction: "What did she play [*NP* Dungeons and ␣]?" is ungrammatical. But you can have the same gap in both branches of a *VP* conjunction, as in the sentence "What did you [*VP* [*VP* smell ␣] and [*VP* shoot an arrow at ␣]]?"

Time and tense: Suppose we want to represent the difference between "Ali loves Bo" and "Ali loved Bo." English uses verb tenses (past, present, and future) to indicate the relative time of an event. One good choice to represent the time of events is the event calculus notation of Section 10.3. In event calculus we have

Time and tense

Ali loves Bo: $E_1 \in Loves(Ali, Bo) \land During(Now, Extent(E_1))$
Ali loved Bo: $E_2 \in Loves(Ali, Bo) \land After(Now, Extent(E_2))$.

This suggests that our two lexical rules for the words "loves" and "loved" should be these:

$$Verb(\lambda y\ \lambda x\ e \in Loves(x,y) \land During(Now, e)) \rightarrow \textbf{loves}$$
$$Verb(\lambda y\ \lambda x\ e \in Loves(x,y) \land After(Now, e)) \rightarrow \textbf{loved}\,.$$

Other than this change, everything else about the grammar remains the same, which is encouraging news; it suggests we are on the right track if we can so easily add a complication like the tense of verbs (although we have just scratched the surface of a complete grammar for time and tense).

Ambiguity

Ambiguity: We tend to think of ambiguity as a failure in communication; when a listener is consciously aware of an ambiguity in an utterance, it means that the utterance is unclear or confusing. Here are some examples taken from newspaper headlines:

> Squad helps dog bite victim.
> Police begin campaign to run down jaywalkers.
> Helicopter powered by human flies.
> Once-sagging cloth diaper industry saved by full dumps.
> Include your children when baking cookies.
> Portable toilet bombed; police have nothing to go on.
> Milk drinkers are turning to powder.
> Two sisters reunited after 18 years in checkout counter.

Such confusions are the exception; most of the time the language we hear seems unambiguous. Thus, when researchers first began to use computers to analyze language in the 1960s, they were quite surprised to learn that almost every sentence is ambiguous, with multiple possible parses (sometimes hundreds), even when the single preferred parse is the only one that native speakers notice. For example, we understand the phrase "brown rice and black beans" as "[brown rice] and [black beans]," and never consider the low-probability interpretation "brown [rice and black beans]," where the adjective "brown" is modifying the whole phrase, not just the "rice." When we hear "Outside of a dog, a book is a person's best friend," we interpret "outside of" as meaning "except for," and find it funny when the next sentence of the Groucho Marx joke is "Inside of a dog it's too dark to read."

Lexical ambiguity

Lexical ambiguity is when a word has more than one meaning: "back" can be an adverb (go back), an adjective (back door), a noun (the back of the room), a verb (back a candidate), or a proper noun (a river in Nunavut, Canada). "Jack" can be a proper name, a noun (a playing card, a six-pointed metal game piece, a nautical flag, a fish, a bird, a cheese, a socket, etc.), or a verb (to jack up a car, to hunt with a light, or to hit a baseball hard). **Syntactic ambiguity** refers to a phrase that has multiple parses: "I smelled a wumpus in 2,2" has two parses: one where the prepositional phrase "in 2,2" modifies the noun and one where it modifies the verb. The syntactic ambiguity leads to a **semantic ambiguity**, because one parse means that the wumpus is in 2,2 and the other means that a stench is in 2,2. In this case, getting the wrong interpretation could be a deadly mistake.

Syntactic ambiguity

Semantic ambiguity

There can also be ambiguity between literal and figurative meanings. Figures of speech are important in poetry, and are common in everyday speech as well. A **metonymy** is a figure of speech in which one object is used to stand for another. When we hear "Chrysler announced a new model," we do not interpret it as saying that companies can talk; rather we understand that a spokesperson for the company made the announcement. Metonymy is common and is often interpreted unconsciously by human hearers.

Metonymy

Unfortunately, our grammar as it is written is not so facile. To handle the semantics of metonymy properly, we need to introduce a whole new level of ambiguity. We could do this by providing *two* objects for the semantic interpretation of every phrase in the sentence: one for the object that the phrase literally refers to (Chrysler) and one for the metonymic reference (the spokesperson). We then have to say that there is a relation between the two. In our current grammar, “Chrysler announced” gets interpreted as

$$x = Chrysler \land e \in Announce(x) \land After(Now, Extent(e)) .$$

We need to change that to

$$x = Chrysler \land e \in Announce(m) \land After(Now, Extent(e)) \\ \land Metonymy(m, x) .$$

This says that there is one entity x that is equal to Chrysler, and another entity m that did the announcing, and that the two are in a metonymy relation. The next step is to define what kinds of metonymy relations can occur. The simplest case is when there is no metonymy at all—the literal object x and the metonymic object m are identical:

$$\forall m, x \;\; (m = x) \;\Rightarrow\; Metonymy(m, x) .$$

For the Chrysler example, a reasonable generalization is that an organization can be used to stand for a spokesperson of that organization:

$$\forall m, x \;\; x \in Organizations \land Spokesperson(m, x) \;\Rightarrow\; Metonymy(m, x) .$$

Other metonymies include the author for the works (I read *Shakespeare*) or more generally the producer for the product (I drive a *Honda*) and the part for the whole (The Red Sox need a strong *arm*). Some examples of metonymy, such as “The *ham sandwich* on Table 4 wants another beer,” are more novel and are interpreted with respect to a situation (such as waiting on tables and not knowing a customer’s name).

A **metaphor** is another figure of speech, in which a phrase with one literal meaning is used to suggest a different meaning by way of an analogy. Thus, metaphor can be seen as a kind of metonymy where the relation is one of similarity. Metaphor

Disambiguation is the process of recovering the most probable intended meaning of an utterance. In one sense we already have a framework for solving this problem: each rule has a probability associated with it, so the probability of an interpretation is the product of the probabilities of the rules that led to the interpretation. Unfortunately, the probabilities reflect how common the phrases are in the corpus from which the grammar was learned, and thus reflect general knowledge, not specific knowledge of the current situation. To do disambiguation properly, we need to combine four models: Disambiguation

1. The **world model**: the likelihood that a proposition occurs in the world. Given what we know about the world, it is more likely that a speaker who says “I’m dead” means “I am in big trouble” or “I lost this video game” rather than “My life ended, and yet I can still talk.”
2. The **mental model**: the likelihood that the speaker forms the intention of communicating a certain fact to the hearer. This approach combines models of what the speaker believes, what the speaker believes the hearer believes, and so on. For example, when a politician says, “I am not a crook,” the world model might assign a probability of only 50% to the proposition that the politician is not a criminal, and 99.999% to the proposition that he is not a hooked shepherd’s staff. Nevertheless, we select the former interpretation because it is a more likely thing to say.
3. The **language model**: the likelihood that a certain string of words will be chosen, given that the speaker has the intention of communicating a certain fact.

4. The **acoustic model**: for spoken communication, the likelihood that a particular sequence of sounds will be generated, given that the speaker has chosen a given string of words. (For handwritten or typed communication, we have the problem of optical character recognition.)

24.6 Natural Language Tasks

Natural language processing is a big field, deserving an entire textbook or two of its own (Goldberg, 2017; Jurafsky and Martin, 2020). In this section we briefly describe some of the main tasks; you can use the references to get more details.

Speech recognition

Speech recognition is the task of transforming spoken sound into text. We can then perform further tasks (such as question answering) on the resulting text. Current systems have a word error rate of about 3% to 5% (depending on details of the test set), similar to human transcribers. The challenge for a system using speech recognition is to respond appropriately even when there are errors on individual words.

Top systems today use a combination of recurrent neural networks and hidden Markov models (Hinton *et al.*, 2012; Yu and Deng, 2016; Deng, 2016; Chiu *et al.*, 2017; Zhang *et al.*, 2017). The introduction of deep neural nets for speech in 2011 led to an immediate and dramatic improvement of about 30% in error rate—this from a field that seemed to be mature and was previously progressing at only a few percent per year. Deep neural networks are a good fit because the problem of speech recognition has a natural compositional breakdown: waveforms to phonemes to words to sentences. They will be covered in the next chapter.

Text-to-speech

Text-to-speech synthesis is the inverse process—going from text to sound. Taylor (2009) gives a book-length overview. The challenge is to pronounce each word correctly, and to make the flow of each sentence seem natural, with the right pauses and emphasis.

Another area of development is in synthesizing different voices—starting with a choice between a generic male or female voice, then allowing for regional dialects, and even imitating celebrity voices. As with speech recognition, the introduction of deep recurrent neural networks led to a large improvement, with about 2/3 of listeners saying that the neural WaveNet system (van den Oord *et al.*, 2016a) sounded more natural than the previous non-neural system.

Machine translation transforms text in one language to another. Systems are usually trained using a bilingual corpus: a set of paired documents, where one member of the pair is in, say, English, and the other is in, say, French. The documents do not need to be annotated in any way; the machine translation system learns to align sentences and phrases and then when presented with a novel sentence in one language, can generate a translation to the other.

Systems in the early 2000s used n-gram models, and achieved results that were usually good enough to get across the meaning of a text, but contained syntactic errors in most sentences. One problem was the limit on the length of the n-grams: even with a large limit of 7, it was difficult for information to flow from one end of the sentence to the other. Another problem was that all the information in an n-gram model is at the level of individual words. Such a system could learn that "black cat" translates to "chat noir," but it could not learn the rule that adjectives generally come before the noun in English and after the noun in French.

Recurrent neural sequence-to-sequence models (Sutskever *et al.*, 2015) got around the problem. They could generalize better (because they could use word embeddings rather than *n*-gram counts of specific words) and could form compositional models throughout the various levels of the deep network to effectively pass information along. Subsequent work using the attention-focusing mechanism of the transformer model (Vaswani *et al.*, 2018) increased performance further, and a hybrid model incorporating aspects of both these models does better still, approaching human-level performance on some language pairs (Wu *et al.*, 2016b; Chen *et al.*, 2018).

Information extraction

Information extraction is the process of acquiring knowledge by skimming a text and looking for occurrences of particular classes of objects and for relationships among them. A typical task is to extract instances of addresses from Web pages, with database fields for street, city, state, and zip code; or instances of storms from weather reports, with fields for temperature, wind speed, and precipitation. If the source text is well structured (for example, in the form of a table), then simple techniques such as regular expressions can extract the information (Cafarella *et al.*, 2008). It gets harder if we are trying to extract *all* facts, rather than a specific type (such as weather reports); Banko *et al.* (2007) describe the TEXTRUNNER system that performs extraction over an open, expanding set of relations. For free-form text, techniques include hidden Markov models and rule-based learning systems (as used in TEXTRUNNER and NELL (Never-Ending Language Learning) (Mitchell *et al.*, 2018)). More recent systems use recurrent neural networks, taking advantage of the flexibility of word embeddings. You can find an overview in Kumar (2017).

Information retrieval

Information retrieval is the task of finding documents that are relevant and important for a given query. Internet search engines such as Google and Baidu perform this task billions of times a day. Three good textbooks on the subject are Manning *et al.* (2008), Croft *et al.* (2010), and Baeza-Yates and Ribeiro-Neto (2011).

Question Answering

Question Answering is a different task, in which the query really is a question, such as “Who founded the U.S. Coast Guard?” and the response is not a ranked list of documents but rather an actual answer: “Alexander Hamilton.” There have been question-answering systems since the 1960s that rely on syntactic parsing as discussed in this chapter, but only since 2001 have such systems used Web information retrieval to radically increase their breadth of coverage. Katz (1997) describes the START parser and question answerer. Banko *et al.* (2002) describe ASKMSR, which was less sophisticated in terms of its syntactic parsing ability, but more aggressive in using Web search and sorting through the results. For example, to answer “Who founded the U.S. Coast Guard?” it would search for queries such as [* founded the U.S. Coast Guard] and [the U.S. Coast Guard was founded by *], and then examine the multiple resulting Web pages to pick out a likely response, knowing that the query word “who” suggests that the answer should be a person. The Text REtrieval Conference (TREC) gathers research on this topic and has hosted competitions on an annual basis since 1991 (Allan *et al.*, 2017). Recently we have seen other test sets, such as the AI2 ARC test set of basic science questions (Clark *et al.*, 2018).

Summary

The main points of this chapter are as follows:

- Probabilistic language models based on *n*-grams recover a surprising amount of information about a language. They can perform well on such diverse tasks as language identification, spelling correction, sentiment analysis, genre classification, and named-entity recognition.

- These language models can have millions of features, so preprocessing and smoothing the data to reduce noise is important.
- In building a statistical language system, it is best to devise a model that can make good use of available **data**, even if the model seems overly simplistic.
- Word embeddings can give a richer representation of words and their similarities.
- To capture the hierarchical structure of language, **phrase structure** grammars (and in particular, **context-free** grammars) are useful. The probabilistic context-free grammar (PCFG) formalism is widely used, as is the dependency grammar formalism.
- Sentences in a context-free language can be parsed in $O(n^3)$ time by a **chart parser** such as the **CYK algorithm**, which requires grammar rules to be in **Chomsky Normal Form**. With a small loss in accuracy, natural languages can be parsed in $O(n)$ time, using a beam search or a shift-reduce parser.
- A **treebank** can be a resource for learning a PCFG grammar with parameters.
- It is convenient to **augment** a grammar to handle issues such as subject–verb agreement and pronoun case, and to represent information at the level of words rather than just at the level of categories.
- **Semantic interpretation** can also be handled by an augmented grammar. We can learn a semantic grammar from a corpus of questions paired either with the logical form of the question, or with the answer.
- Natural language is complex and difficult to capture in a formal grammar.

Bibliographical and Historical Notes

N-gram letter models for language modeling were proposed by Markov (1913). Claude Shannon (Shannon and Weaver, 1949) was the first to generate n-gram word models of English. The **bag-of-words model** gets its name from a passage from linguist Zellig Harris (1954), “language is not merely a bag of words but a tool with particular properties.” Norvig (2009) gives some examples of tasks that can be accomplished with n-gram models.

Chomsky (1956, 1957) pointed out the limitations of finite-state models compared with context-free models, concluding, “Probabilistic models give no particular insight into some of the basic problems of syntactic structure.” This is true, but probabilistic models *do* provide insight into some *other* basic problems—problems that context-free models ignore. Chomsky’s remarks had the unfortunate effect of scaring many people away from statistical models for two decades, until these models reemerged for use in the field of speech recognition (Jelinek, 1976), and in cognitive science, where **optimality theory** (Smolensky and Prince, 1993; Kager, 1999) posited that language works by finding the most probable candidate that optimally satisfies competing constraints.

Add-one smoothing, first suggested by Pierre-Simon Laplace (1816), was formalized by Jeffreys (1948). Other smoothing techniques include interpolation smoothing (Jelinek and Mercer, 1980), Witten–Bell smoothing (1991), Good–Turing smoothing (Church and Gale, 1991), Kneser–Ney smoothing (1995, 2004), and stupid backoff (Brants *et al.*, 2007). Chen and Goodman (1996) and Goodman (2001) survey smoothing techniques.

Simple *n*-gram letter and word models are not the only possible probabilistic models. The **latent Dirichlet allocation** model (Blei *et al.*, 2002; Hoffman *et al.*, 2011) is a probabilistic text model that views a document as a mixture of topics, each with its own distribution of words. This model can be seen as an extension and rationalization of the **latent semantic indexing** model of Deerwester *et al.* (1990) and is also related to the multiple-cause mixture model of (Sahami *et al.*, 1996). And of course there is great interest in non-probabilistic language models, such as the deep learning models covered in Chapter 25.

Joulin *et al.* (2016) give a bag of tricks for efficient text classification. Joachims (2001) uses statistical learning theory and support vector machines to give a theoretical analysis of when classification will be successful. Apté *et al.* (1994) report an accuracy of 96% in classifying Reuters news articles into the "Earnings" category. Koller and Sahami (1997) report accuracy up to 95% with a naive Bayes classifier, and up to 98.6% with a Bayes classifier.

Schapire and Singer (2000) show that simple linear classifiers can often achieve accuracy almost as good as more complex models, and run faster. Zhang *et al.* (2016) describe a character-level (rather than word-level) text classifier. Witten *et al.* (1999) describe compression algorithms for classification, and show the deep connection between the LZW compression algorithm and maximum-entropy language models.

Wordnet (Fellbaum, 2001) is a publicly available dictionary of about 100,000 words and phrases, categorized into parts of speech and linked by semantic relations such as synonym, antonym, and part-of. Charniak (1996) and Klein and Manning (2001) discuss parsing with treebank grammars. The British National Corpus (Leech *et al.*, 2001) contains 100 million words, and the World Wide Web contains several trillion words; Franz and Brants (2006) describe the publicly available Google *n*-gram corpus of 13 million unique words from a trillion words of Web text. Buck *et al.* (2014) describe a similar data set from the Common Crawl project. The Penn Treebank (Marcus *et al.*, 1993; Bies *et al.*, 2015) provides parse trees for a 3-million-word corpus of English.

Many of the *n*-gram model techniques are also used in bioinformatics problems. Biostatistics and probabilistic NLP are coming closer together, as each deals with long, structured sequences chosen from an alphabet.

Early part-of-speech (POS) taggers used a variety of techniques, including rule sets (Brill, 1992), *n*-grams (Church, 1988), decision trees (Màrquez and Rodríguez, 1998), HMMs (Brants, 2000), and logistic regression (Ratnaparkhi, 1996). Historically, a logistic regression model was also called a "maximum entropy Markov model" or MEMM, so some work is under that name. Jurafsky and Martin (2020) have a good chapter on POS tagging. Ng and Jordan (2002) compare discriminative and generative models for classification tasks.

Like semantic networks, context-free grammars were first discovered by ancient Indian grammarians (especially Panini, ca. 350 BCE) studying Shastric Sanskrit (Ingerman, 1967). They were reinvented by Noam Chomsky (1956) for the analysis of English and independently by John Backus (1959) and Peter Naur for the analysis of Algol-58.

Probabilistic context-free grammars were first investigated by Booth (1969) and Salomaa (1969). Algorithms for PCFGs are presented in the excellent short monograph by Charniak (1993) and the excellent long textbooks by Manning and Schütze (1999) and Jurafsky and Martin (2020). Baker (1979) introduces the inside–outside algorithm for learning a

PCFG. **Lexicalized PCFGs** (Charniak, 1997; Hwa, 1998) combine the best aspects of PCFGs and *n*-gram models. Collins (1999) describes PCFG parsing that is lexicalized with head features, and Johnson (1998) shows how the accuracy of a PCFG depends on the structure of the treebank from which its probabilities were learned.

There have been many attempts to write formal grammars of natural languages, both in "pure" linguistics and in computational linguistics. There are several comprehensive but informal grammars of English (Quirk *et al.*, 1985; McCawley, 1988; Huddleston and Pullum, 2002). Since the 1980s, there has been a trend toward lexicalization: putting more information in the lexicon and less in the grammar.

Lexical-functional grammar, or LFG (Bresnan, 1982) was the first major grammar formalism to be highly lexicalized. If we carry lexicalization to an extreme, we end up with **categorial grammar** (Clark and Curran, 2004), in which there can be as few as two grammar rules, or with **dependency grammar** (Smith and Eisner, 2008; Kübler *et al.*, 2009) in which there are no syntactic categories, only relations between words.

Computerized parsing was first demonstrated by Yngve (1955). Efficient algorithms were developed in the 1960s, with a few twists since then (Kasami, 1965; Younger, 1967; Earley, 1970; Graham *et al.*, 1980). Church and Patil (1982) describe syntactic ambiguity and address ways to resolve it.

Klein and Manning (2003) describe A* parsing, and Pauls and Klein (2009) extend that to *K*-best A* parsing, in which the result is not a single parse but the *K* best. Goldberg *et al.* (2013) describe the necessary implementation tricks to make sure that a beam search parser is $O(n)$ and not $O(n^2)$. Zhu *et al.* (2013) describe a fast deterministic shift-reduce parser for natural languages, and Sagae and Lavie (2006) show how adding search to a shift-reduce parser can make it more accurate, at the cost of some speed.

Today, highly accurate open-source parsers include Google's Parsey McParseface (Andor *et al.*, 2016), the Stanford Parser (Chen and Manning, 2014), the Berkeley Parser (Kitaev and Klein, 2018), and the SPACY parser. They all do generalization through neural networks and achieve roughly 95% accuracy on Wall Street Journal or Penn Treebank test sets. There is some criticism of the field that it is focusing too narrowly on measuring performance on a few select corpora, and perhaps overfitting on them.

Formal semantic interpretation of natural languages originates within philosophy and formal logic, particularly Alfred Tarski's (1935) work on the semantics of formal languages. Bar-Hillel (1954) was the first to consider the problems of pragmatics (such as indexicals) and propose that they could be handled by formal logic. Richard Montague's essay "English as a formal language" (1970) is a kind of manifesto for the logical analysis of language, but there are other books that are more readable (Dowty *et al.*, 1991; Portner and Partee, 2002; Cruse, 2011). While semantic interpretation programs are designed to pick the most likely interpretation, literary critics (Empson, 1953; Hobbs, 1990) have been ambiguous about whether ambiguity is something to be resolved or cherished. Norvig (1988) discusses the problems of considering multiple simultaneous interpretations, rather than settling for a single maximum-likelihood interpretation. Lakoff and Johnson (1980) give an engaging analysis and catalog of common metaphors in English. Martin (1990) and Gibbs (2006) offer computational models of metaphor interpretation.

The first NLP system to solve an actual task was the BASEBALL question answering system (Green *et al.*, 1961), which handled questions about a database of baseball statistics.

Close after that was Winograd's (1972) SHRDLU, which handled questions and commands about a blocks-world scene, and Woods's (1973) LUNAR, which answered questions about the rocks brought back from the moon by the Apollo program.

Banko *et al.* (2002) present the ASKMSR question-answering system; a similar system is due to Kwok *et al.* (2001). Pasca and Harabagiu (2001) discuss a contest-winning question-answering system.

Modern approaches to semantic interpretation usually assume that the mapping from syntax to semantics will be learned from examples (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Zhao and Huang, 2015). The first important result on **grammar induction** was a negative one: Gold (1967) showed that it is not possible to reliably learn an exactly correct context-free grammar, given a set of strings from that grammar. Prominent linguists, such as Chomsky (1957) and Pinker (2003), have used Gold's result to argue that there must be an innate **universal grammar** that all children have from birth. The so-called **Poverty of the Stimulus** argument says that children aren't given enough input to learn a CFG, so they must already "know" the grammar and be merely tuning some of its parameters.

Universal grammar

While this argument continues to hold sway throughout much of Chomskyan linguistics, it has been dismissed by other linguists (Pullum, 1996; Elman *et al.*, 1997) and most computer scientists. As early as 1969, Horning showed that it *is* possible to learn, in the sense of PAC learning, a *probabilistic* context-free grammar. Since then, there have been many convincing empirical demonstrations of language learning from positive examples alone, such as learning semantic grammars with inductive logic programming (Muggleton and De Raedt, 1994; Mooney, 1999), the Ph.D. theses of Schütze (1995) and de Marcken (1996), and the entire line of modern language processing systems based on the transformer model (Section 25.4). There is an annual International Conference on Grammatical Inference (ICGI).

James Baker's DRAGON system (Baker, 1975) could be considered the first succesful speech recognition system. It was the first to use HMMs for speech. After several decades of systems based on probabilistic language models, the field began to switch to deep neural networks (Hinton *et al.*, 2012). Deng (2016) describes how the introduction of deep learning enabled rapid improvement in speech recognition, and reflects on the implications for other NLP tasks. Today deep learning is the dominant approach for all large-scale speech recognition systems. Speech recognition can be seen as the first application area that highlighted the success of deep learning, with computer vision following shortly thereafter.

Interest in the field of **information retrieval** was spurred by widespread usage of Internet searching. Croft *et al.* (2010) and Manning *et al.* (2008) provide textbooks that cover the basics. The TREC conference hosts an annual competition for IR systems and publishes proceedings with results.

Brin and Page (1998) describe the PageRank algorithm, which takes into account the links between pages, and give an overview of the implementation of a Web search engine. Silverstein *et al.* (1998) investigate a log of a billion Web searches. The journal *Information Retrieval* and the proceedings of the annual flagship *SIGIR* conference cover recent developments in the field.

Information extraction has been pushed forward by the annual Message Understanding Conferences (MUC), sponsored by the U.S. government. Surveys of template-based systems are given by Roche and Schabes (1997), Appelt (1999), and Muslea (1999). Large databases of facts were extracted by Craven *et al.* (2000), Pasca *et al.* (2006), Mitchell (2007), and

Durme and Pasca (2008). Freitag and McCallum (2000) discuss HMMs for Information Extraction. Conditional random fields have also been used for this task (Lafferty *et al.*, 2001; McCallum, 2003); a tutorial with practical guidance is given by Sutton and McCallum (2007). Sarawagi (2007) gives a comprehensive survey.

Two early influential approaches to automated knowledge engineering for NLP were by Riloff (1993), who showed that an automatically constructed dictionary performed almost as well as a carefully handcrafted domain-specific dictionary, and by Yarowsky (1995), who showed that the task of word sense classification could be accomplished through unsupervised training on a corpus of unlabeled text with accuracy as good as supervised methods.

The idea of simultaneously extracting templates and examples from a handful of labeled examples was developed independently and simultaneously by Blum and Mitchell (1998), who called it **cotraining**, and by Brin (1998), who called it DIPRE (Dual Iterative Pattern Relation Extraction). You can see why the term *cotraining* has stuck. Similar early work, under the name of bootstrapping, was done by Jones *et al.* (1999). The method was advanced by the QXTRACT (Agichtein and Gravano, 2003) and KNOWITALL (Etzioni *et al.*, 2005) systems. Machine reading was introduced by Mitchell (2005) and Etzioni *et al.* (2006) and is the focus of the TEXTRUNNER project (Banko *et al.*, 2007; Banko and Etzioni, 2008).

This chapter has focused on natural language sentences, but it is also possible to do information extraction based on the physical structure or geometric layout of text rather than on the linguistic structure. Lists, tables, charts, graphs, diagrams, etc., whether encoded in HTML or accessed through the visual analysis of pdf documents, are home to data that can be extracted and consolidated (Hurst, 2000; Pinto *et al.*, 2003; Cafarella *et al.*, 2008).

Ken Church (2004) shows that natural language research has cycled between concentrating on the data (empiricism) and concentrating on theories (rationalism); he describes the advantages of having good language resources and evaluation schemes, but wonders if we have gone too far (Church and Hestness, 2019). Early linguists concentrated on actual language usage data, including frequency counts. Noam Chomsky (1956) demonstrated the limitations of finite-state models, leading to an emphasis on theoretical studies of syntax, disregarding actual language performance. This approach dominated for twenty years, until empiricism made a comeback based on the success of work in statistical speech recognition (Jelinek, 1976). Today, the emphasis on empirical language data continues, and there is heightened interest in models that consider higher-level constructs, such as syntactic and semantic relations, not just sequences of words. There is also a strong emphasis on deep learning neural network models of language, which we will cover in Chapter 25.

Work on applications of language processing is presented at the biennial Applied Natural Language Processing conference (ANLP), the conference on Empirical Methods in Natural Language Processing (EMNLP), and the journal *Natural Language Engineering*. A broad range of NLP work appears in the journal *Computational Linguistics* and its conference, ACL, and in the International Computational Linguistics (COLING) conference. Jurafsky and Martin (2020) give a comprehensive introduction to speech and NLP.