11 *Probabilistic Context Free Grammars*

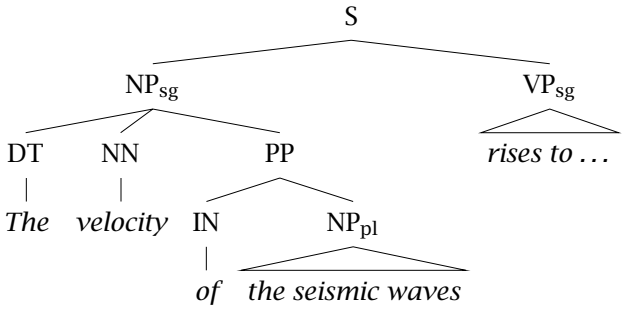
PEOPLE WRITE and say lots of different things, but the way people say things – even in drunken casual conversation – has some structure and regularity. The goal of syntax within linguistics is to try to isolate that structure. Until now, the only form of syntax we have allowed ourselves is methods for describing the ordering and arrangement of words, either directly in terms of the words, or in terms of word categories. In this chapter, we wish to escape the linear tyranny of these *n*-gram models and HMM tagging models, and to start to explore more complex notions of grammar.

Even in the most traditional forms of grammar, syntax is meant to show something more that just linear order. It shows how words group together and relate to each other as heads and dependents. The dominant method used for doing this within the last 50 or so years has been to place tree structures over sentences, as we saw in chapter 3. Language has a complex recursive structure, and such tree-based models – unlike Markov models – allow us to capture this. For instance Kupiec (1992b) notes that his HMM-based tagger has problems with constructions like:

(11.1) The velocity of the seismic waves rises to . . .

because a singular verb (here, *rises*) is unexpected after a plural noun. Kupiec's solution is to augment the HMM model so it can recognize a rudimentary amount of NP structure, which in his model is encoded as a higher-order context extension glued on to his basic first order HMM. Leaving aside the technical details of the solution, the essential observation about verb agreement is that it is reflecting the hierarchical structure of the sentence, as shown in (11.2), and not the linear order of words.

(11.2)



The verb agrees in number with the noun *velocity* which is the head of the preceding noun phrase, and not with the noun that linearly precedes it.

PCFG

PROBABILISTIC CONTEXT FREE GRAMMAR

The simplest probabilistic model for recursive embedding is a *PCFG*, a *Probabilistic* (sometimes also called *Stochastic*) *Context Free Grammar* – which is simply a CFG with probabilities added to the rules, indicating how likely different rewritings are. We provide a detailed discussion of PCFGs in this chapter for a number of reasons: PCFGs are the simplest and most natural probabilistic model for tree structures, the mathematics behind them is well understood, the algorithms for them are a natural development of the algorithms employed with HMMs, and PCFGs provide a sufficiently general computational device that they can simulate various other forms of probabilistic conditioning (as we describe in section 12.1.9). Nevertheless, it is important to realize that PCFGs are only one of many ways of building probabilistic models of syntactic structure, and in the next chapter we study the domain of probabilistic parsing more generally.

A PCFG G consists of:

- A set of terminals, $\{w^k\}, k = 1, \ldots, V$
- A set of nonterminals, $\{N^i\}, i = 1, \ldots, n$
- A designated start symbol, N^1
- A set of rules, $\{N^i \rightarrow \zeta^j\}$, (where ζ^j is a sequence of terminals and nonterminals)
- A corresponding set of probabilities on rules such that:

(11.3) $$\forall i \quad \sum_j P(N^i \rightarrow \zeta^j) = 1$$

Notation	Meaning
G	Grammar (PCFG)
$\mathcal{L}$	Language (generated or accepted by a grammar)
t	Parse tree
$\{N^1, \ldots, N^n\}$	Nonterminal vocabulary (N^1 is start symbol)
$\{w^1, \ldots, w^V\}$	Terminal vocabulary
$w_1 \cdots w_m$	Sentence to be parsed
N^j_{pq}	Nonterminal N^j spans positions p through q in string
$\alpha_j(p,q)$	Outside probabilities (11.15)
$\beta_j(p,q)$	Inside probabilities (11.14)

Table 11.1 Notation for the PCFG chapter.

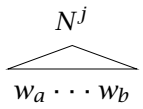
Note that when we write $P(N^i \rightarrow \zeta^j)$ in this chapter, we always mean $P(N^i \rightarrow \zeta^j | N^i)$. That is, we are giving the probability distribution of the daughters for a certain head. Such a grammar can be used either to parse or generate sentences of the language, and we will switch between these terminologies quite freely.

Before parsing sentences with a PCFG, we need to establish some notation. We will represent the sentence to be parsed as a sequence of words $w_1 \cdots w_m$, and use w_{ab} to denote the subsequence $w_a \cdots w_b$. We denote a single rewriting operation of the grammar by a single arrow $\rightarrow$. If as a result of one or more rewriting operations we are able to rewrite a nonterminal N^j as a sequence of words $w_a \cdots w_b$, then we will say

DOMINATION

that N^j *dominates* the words $w_a \cdots w_b$, and write either $N^j \stackrel{*}{\Longrightarrow} w_a \cdots w_b$ or yield$(N^j) = w_a \cdots w_b$. This situation is illustrated in (11.4): a subtree with root nonterminal N^j dominating all and only the words from $w_a \cdots w_b$ in the string:

(11.4)



To say that a nonterminal N_j spans positions a through b in the string, but not to specify what words are actually contained in this subsequence, we will write N^j_{ab}. This notation is summarized in table 11.1.

The probability of a sentence (according to a grammar G) is given by:

(11.5) $$P(w_{1m}) = \sum_t P(w_{1m}, t) \quad \text{where } t \text{ is a parse tree of the sentence}$$

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → *astronomers*	0.1
VP → V NP	0.7	NP → *ears*	0.18
VP → VP PP	0.3	NP → *saw*	0.04
P → *with*	1.0	NP → *stars*	0.18
V → *saw*	1.0	NP → *telescopes*	0.1

Table 11.2 A simple Probabilistic Context Free Grammar (PCFG). The nonterminals are S, NP, PP, VP, P, V. We adopt the common convention whereby the start symbol N^1 is denoted by S. The terminals are the words in italics. The table shows the grammar rules and their probabilities. The slightly unusual NP rules have been chosen so that this grammar is in Chomsky Normal Form, for use as an example later in the section.

$$= \sum_{\{t:\text{yield}(t)=w_{1m}\}} P(t)$$

Moreover, it is easy to find the probability of a tree in a PCFG model. One just multiplies the probabilities of the rules that built its local subtrees.

Example 1: Assuming the grammar in table 11.2, the sentence *astronomers saw stars with ears* has two parses with probabilities as shown in figure 11.1.

What are the assumptions of this model? The conditions that we need are:

- Place invariance. The probability of a subtree does not depend on where in the string the words it dominates are (this is like time invariance in HMMs):

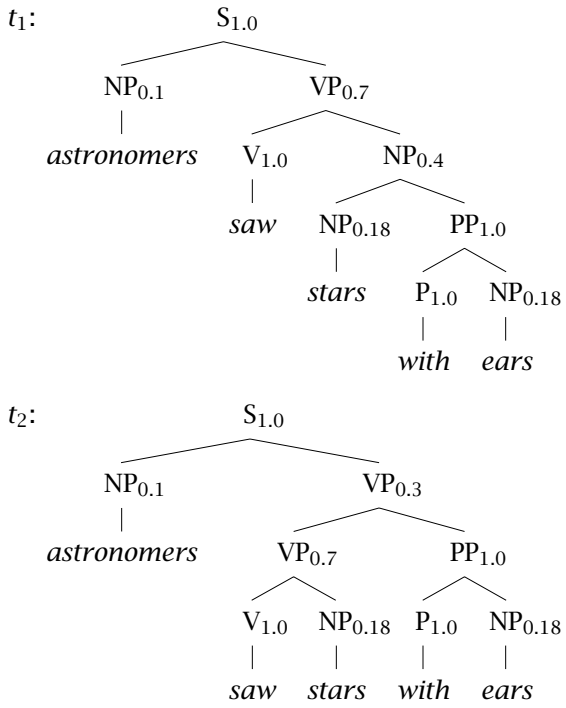
(11.6) $\forall k \quad P(N^j_{k(k+c)} \rightarrow \zeta)$ is the same

- Context-free. The probability of a subtree does not depend on words not dominated by the subtree.

(11.7) $P(N^j_{kl} \rightarrow \zeta | \text{anything outside } k \text{ through } l) = P(N^j_{kl} \rightarrow \zeta)$

- Ancestor-free. The probability of a subtree does not depend on nodes in the derivation outside the subtree.

(11.8) $P(N^j_{kl} \rightarrow \zeta | \text{any ancestor nodes outside } N^j_{kl}) = P(N^j_{kl} \rightarrow \zeta)$



$$
\begin{aligned}
P(t_1) &= 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
&= 0.0009072 \\
P(t_2) &= 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \\
&= 0.0006804 \\
P(w_{15}) &= P(t_1) + P(t_2) = 0.0015876
\end{aligned}
$$

Figure 11.1 The two parse trees, their probabilities, and the sentence probability. This is for the sentence *astronomers saw stars with ears*, according to the grammar in table 11.2. Nonterminal nodes in the trees have been subscripted with the probability of the local tree that they head.

Using these conditions we can justify the calculation of the probability of a tree in terms of just multiplying probabilities attached to rules. But to show an example, we need to be able to distinguish tokens of a nonterminal. Therefore, let the upper left index in ${}^{i}N^{j}$ be an arbitrary identifying index for a particular token of a nonterminal. Then,

$$
\begin{aligned}
P\left(\begin{array}{c} {}^{1}\mathrm{S} \\ {}^{2}\mathrm{NP} \quad {}^{3}\mathrm{VP} \\ \textit{the} \quad \textit{man} \quad \textit{snores} \end{array}\right) &= P({}^{1}\mathrm{S}_{13} \rightarrow {}^{2}\mathrm{NP}_{12}\ {}^{3}\mathrm{VP}_{33}, {}^{2}\mathrm{NP}_{12} \rightarrow \textit{the}_1\ \textit{man}_2, {}^{3}\mathrm{VP}_{33} \rightarrow \textit{snores}_3) \\
&= P({}^{1}\mathrm{S}_{13} \rightarrow {}^{2}\mathrm{NP}_{12}\ {}^{3}\mathrm{VP}_{33}) P({}^{2}\mathrm{NP}_{12} \rightarrow \textit{the}_1\ \textit{man}_2 | {}^{1}\mathrm{S}_{13} \rightarrow {}^{2}\mathrm{NP}_{12}\ {}^{3}\mathrm{VP}_{33}) \\
&\quad P({}^{3}\mathrm{VP}_{33} \rightarrow \textit{snores}_3 | {}^{1}\mathrm{S}_{13} \rightarrow {}^{2}\mathrm{NP}_{12}\ {}^{3}\mathrm{VP}_{33}, {}^{2}\mathrm{NP}_{12} \rightarrow \textit{the}_1\ \textit{man}_2) \\
&= P({}^{1}\mathrm{S}_{13} \rightarrow {}^{2}\mathrm{NP}_{12}\ {}^{3}\mathrm{VP}_{33}) P({}^{2}\mathrm{NP}_{12} \rightarrow \textit{the}_1\ \textit{man}_2) P({}^{3}\mathrm{VP}_{33} \rightarrow \textit{snores}_3) \\
&= P(\mathrm{S} \rightarrow \mathrm{NP}\ \mathrm{VP}) P(\mathrm{NP} \rightarrow \textit{the man}) P(\mathrm{VP} \rightarrow \textit{snores})
\end{aligned}
$$

where, after expanding the probability by the chain rule, we impose first the context-freeness assumption, and then the position-invariant assumption.

11.1 Some Features of PCFGs

Here we give some reasons to use a PCFG, and also some idea of their limitations:

- As grammars expand to give coverage of a large and diverse corpus of text, the grammars become increasingly ambiguous. There start to be many structurally different parses for most word sequences. A PCFG gives some idea of the plausibility of different parses.
- A PCFG does not give a very good idea of the plausibility of different parses, since its probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence.

GRAMMAR INDUCTION
IDENTIFICATION IN THE LIMIT
NEGATIVE EVIDENCE

- PCFGs are good for *grammar induction*. Gold (1967) showed that CFGs cannot be learned (in the sense of *identification in the limit* – that is, whether one can identify a grammar if one is allowed to see as much data produced by the grammar as one wants) without the use of *negative evidence* (the provision of ungrammatical examples). But PCFGs

can be learned from positive data alone (Horning 1969). (However, doing grammar induction from scratch is still a difficult, largely unsolved problem, and hence much emphasis has been placed on learning from bracketed corpora, as we will see in chapter 12.)

- Robustness. Real text tends to have grammatical mistakes, disfluencies, and errors. This problem can be avoided to some extent with a PCFG by ruling out nothing in the grammar, but by just giving implausible sentences a low probability.
- PCFGs give a probabilistic language model for English (whereas a CFG does not).
- The predictive power of a PCFG as measured by entropy tends to be greater than that for a finite state grammar (i.e., an HMM) with the same number of parameters. (For such comparisons, we compute the number of parameters as follows. A V terminal, n nonterminal PCFG has $n^3 + nV$ parameters, while a K state M output HMM has $K^2 + MK$ parameters. While the exponent is higher in the PCFG case, the number of nonterminals used is normally quite small. See Lari and Young (1990) for a discussion of this with respect to certain artificial grammars.)
- In practice, a PCFG is a worse language model for English than an n-gram model (for $n > 1$). An n-gram model takes some local lexical context into account, while a PCFG uses none.
- PCFGs are not good models by themselves, but we could hope to combine the strengths of a PCFG and a trigram model. An early experiment that conditions the rules of a PCFG by word trigrams (and some additional context sensitive knowledge of the tree) is presented in Magerman and Marcus (1991) and Magerman and Weir (1992). Better solutions are discussed in chapter 12.
- PCFGs have certain biases, which may not be appropriate. All else being equal, in a PCFG, the probability of a smaller tree is greater than a larger tree. This is not totally wild – it is consonant with Frazier's (1978) Minimal Attachment heuristic – but it does not give a sensible model of actual sentences, which peak in frequency at some intermediate length. For instance, table 4.3 showed that the most frequent length for *Wall Street Journal* sentences is around 23 words. A PCFG

gives too much of the probability mass to very short sentences. Similarly, all else being equal, nonterminals with a small number of expansions will be favored over nonterminals with many expansions in PCFG parsing, since the individual rewritings will have much higher probability (see exercise 12.3).

The one item here that deserves further comment is the claim that PCFGs define a language model. Initially, one might suspect that providing that the rules all obey equation (11.3), then $\sum_{\omega \in \mathcal{L}} P(\omega) = \sum_t P(t) = 1$. But actually this is only true if the *probability mass of rules* is accumulating in finite derivations. For instance, consider the grammar:

PROBABILITY MASS OF RULES

(11.9)
$$\begin{aligned} &S \rightarrow \textit{rhubarb} && P = \tfrac{1}{3} \\ &S \rightarrow S\ S && P = \tfrac{2}{3} \end{aligned}$$

This grammar will generate all strings *rhubarb … rhubarb*. However, we find that the probability of those strings is:

(11.10)
$$\begin{aligned} &\textit{rhubarb} && \tfrac{1}{3} \\ &\textit{rhubarb rhubarb} && \tfrac{2}{3} \times \tfrac{1}{3} \times \tfrac{1}{3} = \tfrac{2}{27} \\ &\textit{rhubarb rhubarb rhubarb} && \left(\tfrac{2}{3}\right)^2 \times \left(\tfrac{1}{3}\right)^3 \times 2 = \tfrac{8}{243} \end{aligned}$$
…

The probability of the language is the sum of this infinite series $\frac{1}{3} + \frac{2}{27} + \frac{8}{243} + \ldots$, which turns out to be $\frac{1}{2}$. Thus half the probability mass has disappeared into infinite trees which do not generate strings of the language! Such a distribution is often termed *inconsistent* in the probability literature, but since this word has a rather different meaning in other fields related to NLP, we will term such a distribution *improper*. In practice, improper distributions are not much of a problem. Often, it doesn't really matter if probability distributions are improper, especially if we are mainly only comparing the magnitude of different probability estimates. Moreover, providing we estimate our PCFG parameters from parsed training corpora (see chapter 12), Chi and Geman (1998) show that one always gets a proper probability distribution.

INCONSISTENT

IMPROPER

11.2 Questions for PCFGs

Just as for HMMs, there are three basic questions we wish to answer:

- What is the probability of a sentence w_{1m} according to a grammar G: $P(w_{1m}|G)$?
- What is the most likely parse for a sentence: $\arg\max_t P(t|w_{1m}, G)$?
- How can we choose rule probabilities for the grammar G that maximize the probability of a sentence, $\arg\max_G P(w_{1m}|G)$?

CHOMSKY NORMAL FORM

In this chapter, we will only consider the case of *Chomsky Normal Form* grammars, which only have unary and binary rules of the form:

$$N^i \rightarrow N^j \; N^k$$
$$N^i \rightarrow w^j$$

The parameters of a PCFG in Chomsky Normal Form are:

(11.11)
$$P(N^j \rightarrow N^r \; N^s|G) \quad \text{If } n \text{ nonterminals, an } n^3 \text{ matrix of parameters}$$
$$P(N^j \rightarrow w^k|G) \quad \text{If } V \text{ terminals, } nV \text{ parameters}$$

For $j = 1, \ldots, n$,

(11.12)
$$\sum_{r,s} P(N^j \rightarrow N^r \; N^s) + \sum_k P(N^j \rightarrow w^k) = 1$$

This constraint is seen to be satisfied for the grammar in table 11.2 (under the convention whereby all probabilities not shown are zero). Any CFG can be represented by a weakly equivalent CFG in Chomsky Normal Form.[1]

PROBABILISTIC REGULAR GRAMMARS

To see how we might efficiently compute probabilities for PCFGs, let us work from HMMs to *probabilistic regular grammars*, and then from there to PCFGs. Consider a probabilistic regular grammar (PRG), which has rules of the form:

$$N^i \rightarrow w^j \; N^k \quad \text{or} \quad N^i \rightarrow w^j \quad \text{and start state} \quad N^1$$

This is similar to what we had for an HMM. The difference is that in an HMM there is a probability distribution over strings of a certain length:

$$\forall n \quad \sum_{w_{1n}} P(w_{1n}) = 1$$

1. Two grammars G_1 and G_2 are *weakly equivalent* if they both generate the same language $\mathcal{L}$ (with the same probabilities on sentences for stochastic equivalence). Two grammars are *strongly equivalent* if they additionally assign sentences the same tree structures (with the same probabilities, for the stochastic case).

Figure 11.2 A Probabilistic Regular Grammar (PRG).

whereas in a PCFG or a PRG, there is a probability distribution over the set of all strings that are in the language $\mathcal{L}$ generated by the grammar:

$$\sum_{\omega \in \mathcal{L}} P(\omega) = 1$$

To see the difference, consider:

$P(\textit{John decided to bake a})$

This would have a high probability in an HMM, since this is a quite likely beginning to a sentence, but a very low probability in a PRG or a PCFG, because it isn't a complete utterance.

We can think of a PRG as related to an HMM roughly as in figure 11.2. We add a start state and the transitions from it to the states of the HMM mirror the initial probabilities Π. To represent ending the string, we adjoin to the HMM a finish state, often called a *sink state*, which one never leaves once one has entered it. From each HMM state one can continue in the basic HMM or shift to the sink state, which we interpret as the end of string in the PRG.

SINK STATE

This gives the basic idea of how PRGs are related to HMMs. We can implement the PRG as an HMM where the states are nonterminals and the terminals are the output symbols, as follows:

States:	NP	$\longrightarrow$	N′	$\longrightarrow$	N′	$\longrightarrow$	N′	$\longrightarrow$ sink state
	|		|		|		|	
Outputs:	*the*		*big*		*brown*		*box*	

Recall how for an HMM we were able to efficiently do calculations in terms of forward and backward probabilities:

$$\begin{array}{lrcl} \text{Forward probability} & \alpha_i(t) & = & P(w_{1(t-1)}, X_t = i) \\ \text{Backward probability} & \beta_i(t) & = & P(w_{tT} | X_t = i) \end{array}$$

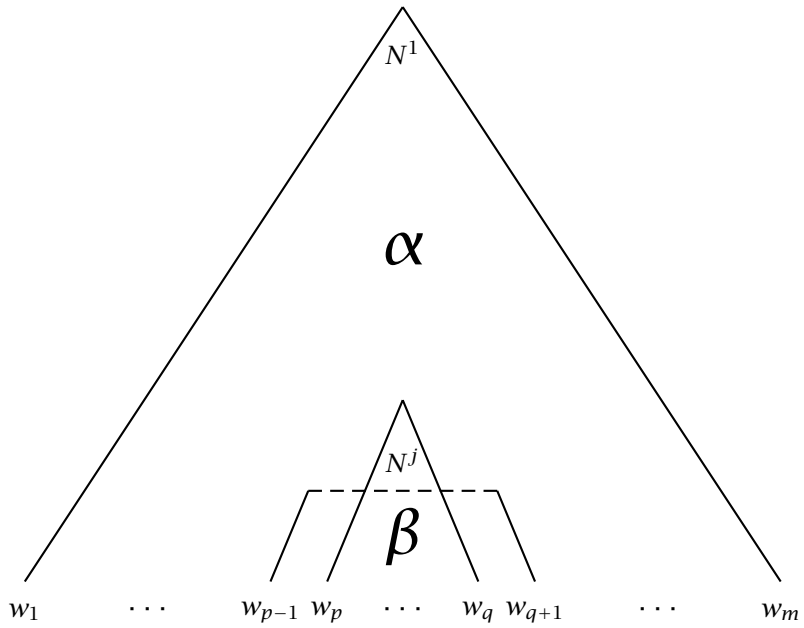
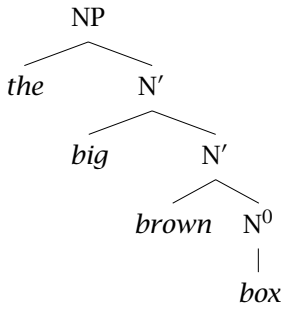


Figure 11.3 Inside and outside probabilities in PCFGs.

But now consider the PRG parse again, drawn as a tree:

(11.13)



In the tree, the forward probability corresponds to the probability of everything above and including a certain node, while the backward probability corresponds to the probability of everything below a certain node (given the node). This suggests an approach to dealing with the more general case of PCFGs. We introduce Inside and Outside probabilities, as indicated in figure 11.3, and defined as follows:

(11.14) Outside probability $\alpha_j(p,q) = P(w_{1(p-1)}, N^j_{pq}, w_{(q+1)m}|G)$

(11.15) Inside probability $\beta_j(p,q) = P(w_{pq}|N^j_{pq}, G)$

The inside probability $\beta_j(p,q)$ is the total probability of generating words $w_p \cdots w_q$ given that one is starting off with the nonterminal N^j. The outside probability $\alpha_j(p,q)$ is the total probability of beginning with the start symbol N^1 and generating the nonterminal N^j_{pq} and all the words outside $w_p \cdots w_q$.

11.3 The Probability of a String

11.3.1 Using inside probabilities

In general, we cannot efficiently calculate the probability of a string by simply summing the probabilities of all possible parse trees for the string, as there will be exponentially many of them. An efficient way to calculate the total probability of a string is by the *inside algorithm*, a dynamic programming algorithm based on the inside probabilities:

INSIDE ALGORITHM

(11.16) $P(w_{1m}|G) = P(N^1 \overset{*}{\Longrightarrow} w_{1m}|G)$

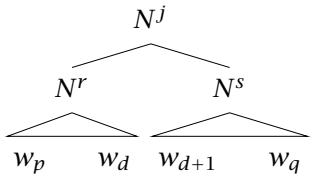
(11.17) $= P(w_{1m}|N^1_{1m}, G) = \beta_1(1,m)$

The inside probability of a substring is calculated by induction on the length of the string subsequence:

Base case: We want to find $\beta_j(k,k)$ (the probability of a rule $N^j \to w_k$):

$$\begin{aligned}\beta_j(k,k) &= P(w_k|N^j_{kk}, G)\\ &= P(N^j \to w_k|G)\end{aligned}$$

Induction: We want to find $\beta_j(p,q)$, for $p < q$. As this is the inductive step using a Chomsky Normal Form grammar, the first rule must be of the form $N^j \to N^r\ N^s$, so we can proceed by induction, dividing the string in two in various places and summing the result:



Then, $\forall j, 1 \leq p < q \leq m$,

$$\begin{aligned}
\beta_j(p,q) &= P(w_{pq}|N^j_{pq}, G) \\
&= \sum_{r,s}\sum_{d=p}^{q-1} P(w_{pd}, N^r_{pd}, w_{(d+1)q}, N^s_{(d+1)q}|N^j_{pq}, G) \\
&= \sum_{r,s}\sum_{d=p}^{q-1} P(N^r_{pd}, N^s_{(d+1)q}|N^j_{pq}, G) P(w_{pd}|N^j_{pq}, N^r_{pd}, N^s_{(d+1)q}, G) \\
&\qquad \times P(w_{(d+1)q}|N^j_{pq}, N^r_{pd}, N^s_{(d+1)q}, w_{pd}, G) \\
&= \sum_{r,s}\sum_{d=p}^{q-1} P(N^r_{pd}, N^s_{(d+1)q}|N^j_{pq}, G) P(w_{pd}|N^r_{pd}, G) \\
&\qquad \times P(w_{(d+1)q}|N^s_{(d+1)q}, G) \\
&= \sum_{r,s}\sum_{d=p}^{q-1} P(N^j \rightarrow N^r N^s) \beta_r(p,d) \beta_s(d+1,q)
\end{aligned}$$

Above, we first divided things up using the chain rule, then we made use of the context-free assumptions of PCFGs, and then rewrote the result using the definition of the inside probabilities. Using this recurrence relation, inside probabilities can be efficiently calculated bottom up.

Example 2: The above equation looks scary, but the calculation of inside probabilities is actually relatively straightforward. We're just trying to find all ways that a certain constituent can be built out of two smaller constituents by varying what the labels of the two smaller constituents are and which words each spans. In table 11.3, we show the computations of inside probabilities using the grammar of table 11.2 and the sentence explored in figure 11.1. The computations are shown using a *parse triangle* where each box records nodes that span from the row index to the column index.

PARSE TRIANGLE

▼ Further calculations using this example grammar and sentence are left to the reader in the Exercises.

	1	2	3	4	5
1	$\beta_{NP} = 0.1$		$\beta_S = 0.0126$		$\beta_S = 0.0015876$
2		$\beta_{NP} = 0.04$ $\beta_V = 1.0$	$\beta_{VP} = 0.126$		$\beta_{VP} = 0.015876$
3			$\beta_{NP} = 0.18$		$\beta_{NP} = 0.01296$
4				$\beta_P = 1.0$	$\beta_{PP} = 0.18$
5					$\beta_{NP} = 0.18$
	astronomers	*saw*	*stars*	*with*	*ears*

Table 11.3 Calculation of inside probabilities. Table cell (p, q) shows non-zero probabilities $\beta^i(p, q)$ calculated via the inside algorithm. The recursive computation of inside probabilities is done starting along the diagonal, and then moving in diagonal rows towards the top right corner. For the simple grammar of table 11.2, the only non-trivial case is cell $(2, 5)$, which we calculate as: $P(\text{VP} \rightarrow \text{V NP})\beta_V(2, 2)\beta_{NP}(3, 5) + P(\text{VP} \rightarrow \text{VP PP})\beta_{VP}(2, 3)\beta_{PP}(4, 5)$

11.3.2 Using outside probabilities

We can also calculate the probability of a string via the use of the outside probabilities. For any k, $1 \leq k \leq m$,

$$
\begin{aligned}
P(w_{1m}|G) &= \sum_j P(w_{1(k-1)}, w_k, w_{(k+1)m}, N^j_{kk}|G) \\
&= \sum_j P(w_{1(k-1)}, N^j_{kk}, w_{(k+1)m}|G) \\
&\qquad \times P(w_k|w_{1(k-1)}, N^j_{kk}, w_{(k+1)n}, G) \qquad (11.18) \\
&= \sum_j \alpha_j(k, k) P(N^j \rightarrow w_k) \qquad (11.19)
\end{aligned}
$$

The outside probabilities are calculated top down. As we shall see, the inductive calculation of outside probabilities requires reference to inside probabilities, so we calculate outside probabilities second, using the *outside algorithm*.

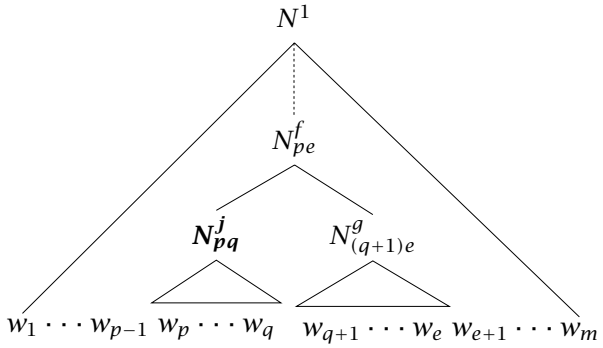
OUTSIDE ALGORITHM

Base Case: The base case is the probability of the root of the tree being nonterminal N^i with nothing outside it:

$$
\begin{aligned}
\alpha_1(1, m) &= 1 \\
\alpha_j(1, m) &= 0 \quad \text{for } j \neq 1
\end{aligned}
$$

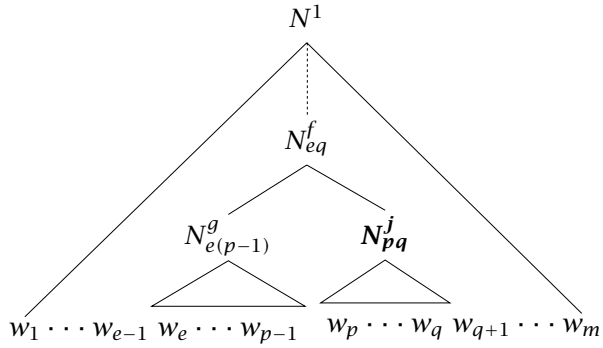
Inductive case: In terms of the previous step of the derivation, a node N^j_{pq} with which we are concerned might be on the left:

(11.20)



or right branch of the parent node:

(11.21)



We sum over both possibilities, but restrict the first sum to $g \neq j$ so as not to double count in the case of rules of the form $X \rightarrow N^j N^j$:

$$\begin{aligned}
\alpha_j(p,q) &= \Big[\sum_{f,g\neq j} \sum_{e=q+1}^{m} P(w_{1(p-1)}, w_{(q+1)m}, N^f_{pe}, N^j_{pq}, N^g_{(q+1)e}) \Big] \\
&\quad + \Big[\sum_{f,g} \sum_{e=1}^{p-1} P(w_{1(p-1)}, w_{(q+1)m}, N^f_{eq}, N^g_{e(p-1)}, N^j_{pq}) \Big] \\
&= \Big[\sum_{f,g\neq j} \sum_{e=q+1}^{m} P(w_{1(p-1)}, w_{(e+1)m}, N^f_{pe}) P(N^j_{pq}, N^g_{(q+1)e} | N^f_{pe}) \\
&\quad \times P(w_{(q+1)e} | N^g_{(q+1)e}) \Big] + \Big[\sum_{f,g} \sum_{e=1}^{p-1} P(w_{1(e-1)}, w_{(q+1)m}, N^f_{eq}) \\
&\quad \times P(N^g_{e(p-1)}, N^j_{pq} | N^f_{eq}) P(w_{e(p-1)} | N^g_{e(p-1)}) \Big]
\end{aligned}$$

$$= \Big[\sum_{f,g \neq j} \sum_{e=q+1}^{m} \alpha_f(p,e) P(N^f \rightarrow N^j \; N^g) \beta_g(q+1,e) \Big] + \Big[\sum_{f,g} \sum_{e=1}^{p-1} \alpha_f(e,q) P(N^f \rightarrow N^g \; N^j) \beta_g(e,p-1) \Big]$$

As with an HMM, we can form a product of the inside and the outside probabilities:

$$\begin{aligned} \alpha_j(p,q)\beta_j(p,q) &= P(w_{1(p-1)}, N^j_{pq}, w_{(q+1)m} | G) P(w_{pq} | N^j_{pq}, G) \\ &= P(w_{1m}, N^j_{pq} | G) \end{aligned}$$

But this time, the fact that we postulate a nonterminal node is important (whereas in the HMM case it is trivial that we are in *some* state at each point in time). Therefore, the probability of the sentence *and* that there is some constituent spanning from word p to q is given by:

$$P(w_{1m}, N_{pq} | G) = \sum_j \alpha_j(p,q)\beta_j(p,q) \tag{11.22}$$

PRETERMINAL

However, we know that there will always be some nonterminal spanning the whole tree and each individual terminal (in a Chomsky Normal Form grammar). The nonterminal nodes whose only daughter is a single terminal node are referred to as *preterminal* nodes. Just in these cases this gives us the equations in (11.17) and (11.19) for the total probability of a string. Equation (11.17) is equivalent to $\alpha_1(1,m)\beta_1(1,m)$ and makes use of the root node, while equation (11.19) is equivalent to $\sum_j \alpha_j(k,k)\beta_j(k,k)$ and makes use of the fact that there must be some preterminal N^j above each word w_k.

11.3.3 Finding the most likely parse for a sentence

A Viterbi-style algorithm for finding the most likely parse for a sentence can be constructed by adapting the inside algorithm to find the element of the sum that is maximum, and to record which rule gave this maximum. This works as in the HMM case because of the independence assumptions of PCFGs. The result is an $O(m^3 n^3)$ PCFG parsing algorithm.

The secret to the Viterbi algorithm for HMMs is to define accumulators $\delta_j(t)$ which record the highest probability of a path through the trellis that leaves us at state j at time t. Recalling the link between HMMs and PCFGs through looking at PRGs, this time we wish to find the highest

probability partial parse tree spanning a certain substring that is rooted with a certain nonterminal. We will retain the name δ and use accumulators:

$\delta_i(p,q)$ = the highest inside probability parse of a subtree N^i_{pq}

Using dynamic programming, we can then calculate the most probable parse for a sentence as follows. The initialization step assigns to each unary production at a leaf node its probability. For the inductive step, we again know that the first rule applying must be a binary rule, but this time we find the most probable one instead of summing over all such rules, and record that most probable one in the ψ variables, whose values are a list of three integers recording the form of the rule application which had the highest probability.

1. Initialization

   $$\delta_i(p,p) = P(N^i \rightarrow w_p)$$

2. Induction

   $$\delta_i(p,q) = \max_{\substack{1 \le j,k \le n \\ p \le r < q}} P(N^i \rightarrow N^j \ N^k)\delta_j(p,r)\delta_k(r+1,q)$$

   Store backtrace

   $$\psi_i(p,q) = \arg\max_{(j,k,r)} P(N^i \rightarrow N^j \ N^k)\delta_j(p,r)\delta_k(r+1,q)$$

3. Termination and path readout (by backtracking). Since our grammar has a start symbol N^1, then by construction, the probability of the most likely parse rooted in the start symbol is:[2]

(11.23) $$P(\hat{t}) = \delta_1(1,m)$$

   We want to reconstruct this maximum probability tree $\hat{t}$. We do this by regarding $\hat{t}$ as a set of nodes $\{\hat{X}_x\}$ and showing how to construct this

2. We could alternatively find the highest probability node of any category that dominates the entire sentence as:

$$P(\hat{t}) = \max_{1 \le i \le n} \delta_i(1,m)$$

set. Since the grammar has a start symbol, the root node of the tree must be N^1_{1m}. We then show in general how to construct the left and right daughter nodes of a nonterminal node, and applying this process recursively will allow us to reconstruct the entire tree. If $X_x = N^i_{pq}$ is in the Viterbi parse, and $\psi_i(p,q) = (j,k,r)$, then:

$$\begin{aligned} \text{left}(\hat{X}_x) &= N^j_{pr} \\ \text{right}(\hat{X}_x) &= N^k_{(r+1)q} \end{aligned}$$

Note that where we have written 'arg max' above, it is possible for there not to be a unique maximum. We assume that in such cases the parser just chooses one maximal parse at random. It actually makes things considerably more complex to preserve all ties.

11.3.4 Training a PCFG

The idea of training a PCFG is grammar learning or grammar induction, but only in a certain limited sense. We assume that the structure of the grammar in terms of the number of terminals and nonterminals, and the name of the start symbol is given in advance. We also assume the set of rules is given in advance. Often one assumes that all possible rewriting rules exist, but one can alternatively assume some pre-given structure in the grammar, such as making some of the nonterminals dedicated preterminals that may only be rewritten as a terminal node. Training the grammar comprises simply a process that tries to find the optimal probabilities to assign to different grammar rules within this architecture.

INSIDE-OUTSIDE ALGORITHM

As in the case of HMMs, we construct an EM training algorithm, the *Inside-Outside algorithm*, which allows us to train the parameters of a PCFG on unannotated sentences of the language. The basic assumption is that a good grammar is one that makes the sentences in the training corpus likely to occur, and hence we seek the grammar that maximizes the likelihood of the training data. We will present training first on the basis of a single sentence, and then show how it is extended to the more realistic situation of a large training corpus of many sentences, by assuming independence between sentences.

To determine the probability of rules, what we would like to calculate is:

$$\hat{P}(N^j \to \zeta) = \frac{C(N^j \to \zeta)}{\sum_\gamma C(N^j \to \gamma)}$$

where $C(\cdot)$ is the count of the number of times that a particular rule is used. If parsed corpora are available, we can calculate these probabilities directly (as discussed in chapter 12). If, as is more common, a parsed training corpus is not available, then we have a hidden data problem: we wish to determine probability functions on rules, but can only directly see the probabilities of sentences. As we don't know the rule probabilities, we cannot compute relative frequencies, so we instead use an iterative algorithm to determine improving estimates. We begin with a certain grammar topology, which specifies how many terminals and nonterminals there are, and some initial probability estimates for rules (perhaps just randomly chosen). We use the probability of each parse of a training sentence according to this grammar as our confidence in it, and then sum the probabilities of each rule being used in each place to give an expectation of how often each rule was used. These expectations are then used to refine our probability estimates on rules, so that the likelihood of the training corpus given the grammar is increased.

Consider:

$$\begin{aligned}\alpha_j(p,q)\beta_j(p,q) &= P(N^1 \overset{*}{\Rightarrow} w_{1m}, N^j \overset{*}{\Rightarrow} w_{pq}|G) \\ &= P(N^1 \overset{*}{\Rightarrow} w_{1m}|G)P(N^j \overset{*}{\Rightarrow} w_{pq}|N^1 \overset{*}{\Rightarrow} w_{1m}, G)\end{aligned}$$

We have already solved how to calculate $P(N^1 \overset{*}{\Rightarrow} w_{1m})$; let us call this probability π. Then:

$$P(N^j \overset{*}{\Rightarrow} w_{pq}|N^1 \overset{*}{\Rightarrow} w_{1m}, G) = \frac{\alpha_j(p,q)\beta_j(p,q)}{\pi}$$

and the estimate for how many times the nonterminal N^j is used in the derivation is:

$$E(N^j \text{ is used in the derivation}) = \sum_{p=1}^{m}\sum_{q=p}^{m}\frac{\alpha_j(p,q)\beta_j(p,q)}{\pi} \tag{11.24}$$

In the case where we are not dealing with a preterminal, we substitute the inductive definition of β into the above probability and then $\forall r, s, p < q$:

$$\begin{aligned}&P(N^j \to N^r\ N^s \overset{*}{\Rightarrow} w_{pq}|N^1 \overset{*}{\Rightarrow} w_{1m}, G) \\ &= \frac{\sum_{d=p}^{q-1}\alpha_j(p,q)P(N^j \to N^r\ N^s)\beta_r(p,d)\beta_s(d+1,q)}{\pi}\end{aligned}$$

Therefore, the estimate for how many times this particular rule is used in the derivation can be found by summing over all ranges of words that

the node could dominate:

$$E(N^j \to N^r\ N^s, N^j \text{ used}) = \frac{\sum_{p=1}^{m-1}\sum_{q=p+1}^{m}\sum_{d=p}^{q-1}\alpha_j(p,q)P(N^j \to N^r\ N^s)\beta_r(p,d)\beta_s(d+1,q)}{\pi} \tag{11.25}$$

Now for the maximization step, we want:

$$P(N^j \to N^r\ N^s) = \frac{E(N^j \to N^r\ N^s, N^j \text{ used})}{E(N^j \text{ used})}$$

So, the reestimation formula is:

$$\hat{P}(N^j \to N^r\ N^s) = (11.25)/(11.24) = \frac{\sum_{p=1}^{m-1}\sum_{q=p+1}^{m}\sum_{d=p}^{q-1}\alpha_j(p,q)P(N^j \to N^r\ N^s)\beta_r(p,d)\beta_s(d+1,q)}{\sum_{p=1}^{m}\sum_{q=p}^{m}\alpha_j(p,q)\beta_j(p,q)} \tag{11.26}$$

Similarly for preterminals,

$$P(N^j \to w^k | N^1 \overset{*}{\Longrightarrow} w_{1m}, G) = \frac{\sum_{h=1}^{m}\alpha_j(h,h)P(N^j \to w_h, w_h = w^k)}{\pi} = \frac{\sum_{h=1}^{m}\alpha_j(h,h)P(w_h = w^k)\beta_j(h,h)}{\pi}$$

The $P(w_h = w^k)$ above is, of course, either 0 or 1, but we express things in the second form to show maximal similarity with the preceding case. Therefore,

$$\hat{P}(N^j \to w^k) = \frac{\sum_{h=1}^{m}\alpha_j(h,h)P(w_h = w^k)\beta_j(h,h)}{\sum_{p=1}^{m}\sum_{q=p}^{m}\alpha_j(p,q)\beta_j(p,q)} \tag{11.27}$$

Unlike the case of HMMs, this time we cannot possibly avoid the problem of dealing with multiple training instances – one cannot use concatenation as in the HMM case. Let us assume that we have a set of training sentences $W = (W_1, \ldots, W_\omega)$, with $W_i = w_{i,1} \cdots w_{i,m_i}$. Let f_i, g_i, and h_i be the common subterms from before for use of a nonterminal at a branching node, at a preterminal node, and anywhere respectively, now calculated from sentence W_i:

$$f_i(p,q,j,r,s) = \frac{\sum_{d=p}^{q-1}\alpha_j(p,q)P(N^j \to N^r N^s)\beta_r(p,d)\beta_s(d+1,q)}{P(N^1 \overset{*}{\Longrightarrow} W_i | G)} \tag{11.28}$$

$$g_i(h,j,k) = \frac{\alpha_j(h,h)P(w_h = w^k)\beta_j(h,h)}{P(N^1 \overset{*}{\Longrightarrow} W_i | G)}$$

$$h_i(p,q,j) = \frac{\alpha_j(p,q)\beta_j(p,q)}{P(N^1 \overset{*}{\Longrightarrow} W_i | G)}$$

If we assume that the sentences in the training corpus are independent, then the likelihood of the training corpus is just the product of the probabilities of the sentences in it according to the grammar. Therefore, in the reestimation process, we can sum the contributions from multiple sentences to give the following reestimation formulas. Note that the denominators consider all expansions of the nonterminal, as terminals or nonterminals, to satisfy the stochastic constraint in equation (11.3) that a nonterminal's expansions sum to 1.

$$\hat{P}(N^j \rightarrow N^r\ N^s) = \frac{\sum_{i=1}^{\omega}\sum_{p=1}^{m_i-1}\sum_{q=p+1}^{m_i} f_i(p,q,j,r,s)}{\sum_{i=1}^{\omega}\sum_{p=1}^{m_i}\sum_{q=p}^{m_i} h_i(p,q,j)} \tag{11.29}$$

and

$$\hat{P}(N^j \rightarrow w^k) = \frac{\sum_{i=1}^{\omega}\sum_{h=1}^{m_i} g_i(h,j,k)}{\sum_{i=1}^{\omega}\sum_{p=1}^{m_i}\sum_{q=p}^{m_i} h_i(p,q,j)} \tag{11.30}$$

The Inside-Outside algorithm is to repeat this process of parameter reestimation until the change in the estimated probability of the training corpus is small. If G_i is the grammar (including rule probabilities) in the i^{th} iteration of training, then we are guaranteed that the probability of the corpus according to the model will improve or at least get no worse:

$$P(W|G_{i+1}) \geq P(W|G_i).$$

11.4 Problems with the Inside-Outside Algorithm

However, the PCFG learning algorithm is not without problems:

1. Compared with linear models like HMMs, it is slow. For each sentence, each iteration of training is $O(m^3n^3)$, where m is the length of the sentence, and n is the number of nonterminals in the grammar.

LOCAL MAXIMA

2. *Local maxima* are much more of a problem. Charniak (1993) reports that on each of 300 trials of PCFG induction (from randomly initialized parameters, using artificial data generated from a simple English-like PCFG) a different local maximum was found. Or in other words, the algorithm is very sensitive to the initialization of the parameters. This might perhaps be a good place to try another learning method. (For instance, the process of simulated annealing has been used with some success with neural nets to avoid problems of getting stuck in local

maxima (Kirkpatrick et al. 1983; Ackley et al. 1985), but it is still perhaps too compute expensive for large-scale PCFGs.) Other partial solutions are restricting rules by initializing some parameters to zero or performing grammar minimization, or reallocating nonterminals away from "greedy" terminals. Such approaches are discussed in Lari and Young (1990).

3. Based on experiments on artificial languages, Lari and Young (1990) suggest that satisfactory grammar learning requires many more nonterminals than are theoretically needed to describe the language at hand. In their experiments one typically needed about $3n$ nonterminals to satisfactorily learn a grammar from a training text generated by a grammar with n nonterminals. This compounds the first problem.

4. While the algorithm is guaranteed to increase the probability of the training corpus, there is no guarantee that the nonterminals that the algorithm learns will have any satisfactory resemblance to the kinds of nonterminals normally motivated in linguistic analysis (NP, VP, etc.). Even if one initializes training with a grammar of the sort familiar to linguists, the training regime may completely change the meaning of nonterminal categories as it thinks best. As we have set things up, the only hard constraint is that N^1 must remain the start symbol. One option is to impose further constraints on the nature of the grammar. For instance, one could specialize the nonterminals so that they each only generate terminals *or* nonterminals. Using this form of grammar would actually also simplify the reestimation equations we presented above.

Thus, while grammar induction from unannotated corpora is possible in principle with PCFGs, in practice, it is extremely difficult. In different ways, many of the approaches of the next chapter address various of the limitations of using vanilla PCFGs.

11.5 Further Reading

A comprehensive discussion of topics like weak and strong equivalence, Chomsky Normal Form, and algorithms for changing arbitrary CFGs into various normal forms can be found in (Hopcroft and Ullman 1979). Standard techniques for parsing with CFGs in NLP can be found in most AI and NLP textbooks, such as (Allen 1995).

Probabilistic CFGs were first studied in the late 1960s and early 1970s, and initially there was an outpouring of work. Booth and Thomson (1973), following on from Booth (1969), define a PCFG as in this chapter (modulo notation). Among other results, they show that there are probability distributions on the strings of context free languages which cannot be generated by a PCFG, and derive necessary and sufficient conditions for a PCFG to define a proper probability distribution. Other work from this period includes: (Grenander 1967), (Suppes 1970), (Huang and Fu 1971), and several PhD theses (Horning 1969; Ellis 1969; Hutchins 1970). Tree structures in probability theory are normally referred to as *branching processes*, and are discussed in such work as (Harris 1963) and (Sankoff 1971).

BRANCHING PROCESSES

During the 1970s, work on stochastic formal languages largely died out, and PCFGs were really only kept alive by the speech community, as an occasionally tried variant model. The Inside-Outside algorithm was introduced, and its convergence properties formally proved by Baker (1979). Our presentation essentially follows (Lari and Young 1990). This paper includes a proof of the algorithmic complexity of the Inside-Outside algorithm. Their work is further developed in (Lari and Young 1991).

For the extension of the algorithms presented here to arbitrary PCFGs, see (Charniak 1993) or (Kupiec 1991, 1992a).[3] Jelinek et al. (1990) and Jelinek et al. (1992a) provide a thorough introduction to PCFGs. In particular, these reports, and also Jelinek and Lafferty (1991) and Stolcke (1995), present incremental left-to-right versions of the Inside and Viterbi algorithms, which are very useful in contexts such as language models for speech recognition.

In the section on training a PCFG, we assumed a fixed grammar architecture. This naturally raises the question of how one should determine this architecture, and how one would learn it automatically. There has been a little work on automatically determining a suitable architecture using *Bayesian model merging*, a *Minimum Description Length* approach (Stolcke and Omohundro 1994b; Chen 1995), but at present this task is still normally carried out by using the intuitions of a linguist.

BAYESIAN MODEL MERGING
MINIMUM DESCRIPTION LENGTH

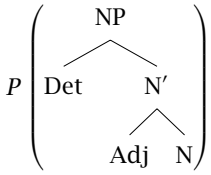
3. For anyone familiar with chart parsing, the extension is fairly straightforward: in a chart we always build maximally binary 'traversals' as we move the dot through rules. We can use this virtual grammar, with appropriate probabilities to parse arbitrary PCFGs (the rule that completes a constituent can have the same probability as the original rule, while all others have probability 1).

PCFGs have also been used in bioinformatics (e.g., Sakakibara et al. 1994), but not nearly as much as HMMs.

11.6 Exercises

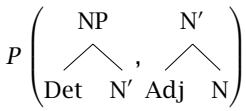
Exercise 11.1 [⋆⋆]

Consider the probability of a (partial) syntactic parse tree giving part of the structure of a sentence:



In general, as the (sub)tree gets large, we cannot accurately estimate the probability of such trees from any existing training corpus (a data sparseness problem).

As we saw, PCFGs approach this problem by estimating the probability of a tree like the one above from the joint probabilities of local subtrees:



However, how reasonable is it to assume independence between the probability distributions of these local subtrees (which is the assumption that licenses us to estimate the probability of a subtree as the product of the probability of each local tree it contains)?

Use a parsed corpus (e.g., the Penn Treebank) and find for some common subtrees whether the independence assumption seems justified or not. If it is not, see if you can find a method of combining the probabilities of local subtrees in such a way that it results in an empirically better estimate of the probability of a larger subtree.

Exercise 11.2 [⋆]

Using a parse triangle as in figure 11.3, calculate the outside probabilities for the sentence *astronomers saw stars with ears* according to the grammar in table 11.2. Start at the top righthand corner and work towards the diagonal.

Exercise 11.3 [⋆]

Using the inside and outside probabilities for the sentence *astronomers saw stars with ears* worked out in figure 11.3 and exercise 11.2, reestimate the probabilities of the grammar in table 11.2 by working through one iteration of the Inside-Outside algorithm. It is helpful to first link up the inside probabilities shown in figure 11.3 with the particular rules and subtrees used to obtain them.

What would the rule probabilities converge to with continued iterations of the Inside-Outside algorithm? Why?

Exercise 11.4 [⋆⋆⋆]

Recording possible spans of nodes in a parse triangle such as the one in figure 11.3 is the essence of the Cocke-Kasami-Younger (CKY) algorithm for parsing CFGs (Younger 1967; Hopcroft and Ullman 1979). Writing a CKY PCFG parser is quite straightforward, and a good exercise. One might then want to extend the parser from Chomsky Normal Form grammars to the more general case of context-free grammars. One way is to work out the general case oneself, or to consult the appropriate papers in the Further Reading. Another way is to write a grammar transformation that will take a CFG and convert it into a Chomsky Normal Form CFG by introducing specially-marked additional nodes where necessary, which can then be removed on output to display parse trees as given by the original grammar. This task is quite easy if one restricts the input CFG to one that does not contain any empty nodes (nonterminals that expand to give nothing).

Exercise 11.5 [⋆⋆⋆]

Rather than simply parsing a sequence of words, if interfacing a parser to a speech recognizer, one often wants to be able to parse a word lattice, of the sort shown in figure 12.1. Extend a PCFG parser so it works with word lattices. (Because the runtime of a PCFG parser is dependent on the number of words in the word lattice, a PCFG parser can be impractical when dealing with large speech lattices, but our CPUs keep getting faster every year!)