

## Context Free Grammars

So far we have looked at models of language that capture only local phenomena, namely what we can analyze when looking at only a small window of words in a sentence. To move towards more sophisticated applications that require some form of understanding, we need to be able to determine larger structures and to make decisions based on information that may be further away than we can capture in any fixed window. Here are some properties of language that we would like to be able to analyze:

1. **Structure and Meaning**—A sentence has a semantic structure that is critical to determine for many tasks. For instance, there is a big difference between *Foxes eat rabbits* and *Rabbits eat foxes*. Both describe some eating event, but in the first the fox is the eater, and in the second it is the one being eaten. We need some way to represent overall sentence structure that captures such relationships.
2. **Agreement**—In English as in most languages, there are constraints between forms of words that should be maintained, and which are useful in analysis for eliminating implausible interpretations. For instance, English requires number agreement between the subject and the verb (among other things). Thus we can say *Foxes eat rabbits* but “Foxes eats rabbits” is ill-formed. In contrast, *The Fox eats a rabbit* is fine whereas “The fox eat a rabbit” is not.
3. **Recursion**—Natural languages have a recursive structure that is important to capture. For instance, *Foxes that eat chickens eat rabbits* is best analyzed as having an embedded sentence (i.e., *that eat chickens*) modifying the noun phases *Foxes*. This one additional rule allows us to also interpret *Foxes that eat chickens that eat corn eat rabbits* and a host of other sentences.
4. **Long Distance Dependencies**—Because of the recursion, word dependencies like number agreement can be arbitrarily far apart from each other, e.g., “Foxes that eat chickens that eat corn eats rabbits” is awkward because the subject (foxes) is plural while the main verb (eats) is singular. In principle, we could find sentences that have an arbitrary number of words between the subject and its verb.

### ***Tree Representations of Structure***

The standard representation of sentence structure to capture these properties is a tree structure as shown in Figure 1 for the sentence *The birds in the field eat the corn*. A **tree** consists of labeled **nodes** with arcs indicates a parent-child relationship. Trees have a unique **root** node (S in figure 1). Every node except the root has exactly one parent node, and 0 or more children. Nodes without children are called the **leaf nodes**. This tree captures the following structural analysis: there is a sentence (S) that consists of an NP followed by a VP. The NP is an article ART followed by a noun (N) and a prepositional phrase (PP). The ART is *the*, the N is *birds* and the PP consists of a proposition (P) followed by an NP. The P is *in*, and the NP consists of an ART (*the*) followed by a N (*field*). Returning to the VP, it consists of a V followed by an NP, and so on through the structure. Note that the tree form readily captures the recursive nature of language. In this tree we have an NP as a subpart of a larger NP. Thus this structure could also easily represent the structure of the sentence *The birds in the field in the forest eat the corn*, where the inner NP itself also contains a PP that contains another NP.

Note also that the structure allows us to state dependencies between constituents in a concise way. For instance, English requires number agreement between the subject NP (however complex) and the main verb.

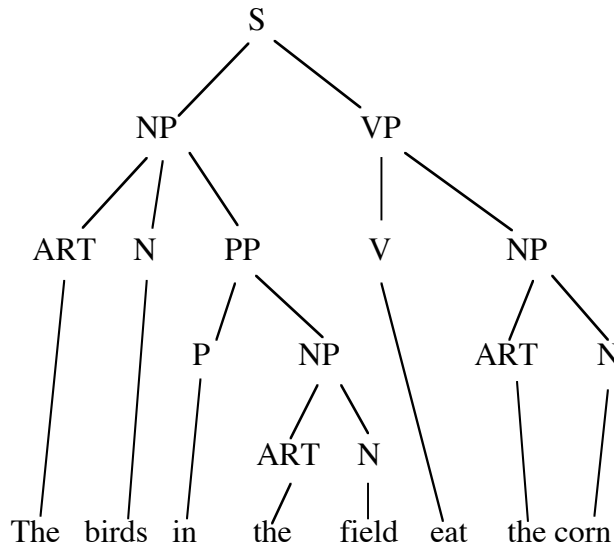


Figure 1: A tree representation of sentence structure

This constraint

is easily captured as additional information on when an NP followed by a VP can produce a reasonable sentence S. To make properties like agreement work in practice would require developing a more general formalism for expressing constraints on trees, which takes us beyond our focus in this course (but see a text on Natural Language Understanding).

### The Context-Free Grammar Formalism

Given a tree analysis, how do we tell what trees characterize acceptable structures in the language? The space of allowable trees is specified by a **Grammar**, in our case a **Context Free Grammar (CFG)**. A CFG consists of a set of the form

$$\text{category}_0 \rightarrow \text{category}_1 \dots \text{category}_n$$

which states that a node labeled with  $\text{category}_0$  is allowed in a tree when it has  $n$  children of categories (from left to right)  $\text{category}_1 \dots \text{category}_n$ . For example, Figure 2 gives a set of rules that would allow all the trees in the previous figures.

You can easily verify that every parent-child relationship shown in each of the trees is listed as a rule in the grammar. Typically, a CFG defines an infinite space of possible trees, and for any given sentence there may be a number of possible trees that are possible. These different trees can provide an account of some of the ambiguities that are inherent in language. For example, Figure 3 provides two trees associated with the sentence *I hate annoying neighbors*. On the left, we have the interpretation that the neighbors are noisy and we don't like them. On the right, we have the interpretation that we don't like doing things (like playing loud music) that annoy our neighbors.

```

S -> NP VP
NP -> PRO
NP -> ART N
NP -> N
NP -> ADJ N
NP -> NP S
VP -> V NP
VP -> V VP
  
```

Figure 2: A context-free grammar

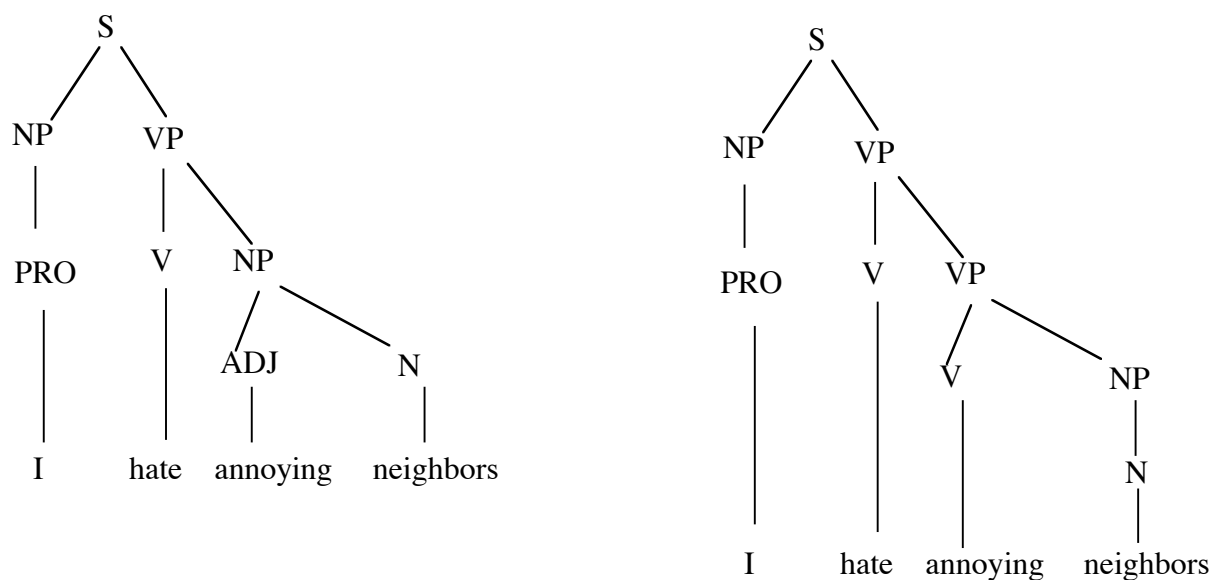


Figure 3: Two tree representation of the same sentence

This ambiguity arises from choosing a different lexical category for the word *annoying*, which then forces different structural analyses. There are many other cases of ambiguity that are purely structural, and the words are interpreted the same way in each case. The classic example of this is the sentence *I saw the man with a telescope*, which is ambiguous between me seeing a man holding a telescope and me using a telescope to see a man. This ambiguity would be reflected in where the prepositional phrase with a telescope attached into the parse tree.

### **Probabilistic Context Free Grammars**

Given a grammar and a sentence, a natural question to ask it 1) whether there is any tree that accounts for the sentence (i.e., is the sentence grammatical), and 2) if there are many trees, which is the right interpretation (i.e., disambiguation). For the first question, we need to be able to build a parse tree for a sentence if one exists. To answer the second, we need to have some way of comparing the likelihood of one tree over another. There have been many attempts to try to find inherent preferences based on the structure of trees. For instance, we might say we prefer tree that are smaller over ones that are larger. But none of these approaches has been able to provide a satisfactory account of parse preferences. Putting this into a probabilistic framework, given a grammar  $G$ , sentence  $s$ , and a set of trees  $T_s$  that could account for  $s$  (i.e, each  $t$  has  $s$  as its leaf nodes), we want the parse tree that is

$$\operatorname{argmax}_{t \in T_s} P_G(t_s)$$

To compute this directly, we would need to know the probability distribution  $P_G(T)$ , where  $T$  is the set of all possible trees. Clearly we will never be able to estimate this distribution directly. As usual, we must make some independence assumptions. The most radical assumption is that each node in the tree is decomposed independently of the rest of the nodes in the tree. In other words,

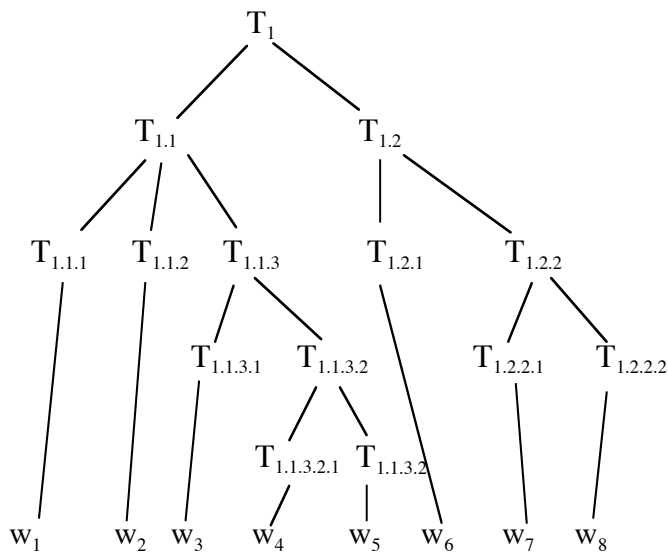


Figure 4: The notation for analyzing trees

say we have a tree  $T$  consisting of nodes  $T_1$ , with children  $T_{1,1}, \dots, T_{1,n}$ , with  $T_{i,j}$  having children  $T_{i,j,1}, \dots, T_{i,j,m}$  and so on, as shown in figure 4.

Thus making the independence assumption about, we'd say

$$P(T) = P(T_1 \rightarrow T_{1,1} T_{1,2}) * P(T_{1,1}) * P(T_{1,2})$$

Using the independence assumptions repeatedly, we can expand  $P(T_{1,1})$  to  $P(T_{1,1} \rightarrow T_{1,1,1} T_{1,1,2} T_{1,1,3}) * P(T_{1,1,1}) * P(T_{1,1,2}) * P(T_{1,1,3})$ , and so on until we have expanded out the probabilities of all the subtrees. Having done this, we'd end up with the following:

$$P(T) = \prod_r P(r) \text{ where } r \text{ ranges over all rules used to construct } T$$

The probability of a rule  $r$  is the probability that the rule will be used to rewrite the node on the right hand side. Thus, if 3/4 of NPs are built by the rule  $NP \rightarrow ART N$ , then the probability of this rule would be .75. This formalism is called a **probabilistic context-free grammar (PCFG)**.

### **Estimating Rule Probabilities**

There are a number of ways to estimate the probabilities for a PCFG. The best, but most labor intensive way is to hand-construct a corpus of parse trees for a set of sentences, and then estimate the probabilities of each rule being used by counting over the corpus. Specifically, the MLE estimate for a rule  $RHS \rightarrow LHS$  would be

$$P_{MLE}(RHS \rightarrow LHS) = \text{Count}(RHS \rightarrow LHS) / \sum_c \text{Count}(RHS \rightarrow c)$$

For example, given the trees in figures 1 and 2 as our training corpus, we would obtain the

estimates for the rule probabilities in figure 5.

Rule	Count	Total for RHS	MLE Estimate
S -> NP VP	3	3	1
NP -> PRO	2	7	.29
NP -> ART N	2	7	.29
NP -> N	1	7	.15
NP -> ADJ N	1	7	.15
NP -> ART ADJ N	1	7	.15
VP -> V NP	3	4	.75
VP -> V VP	1	4	.25
PP -> P NP	1	1	1
PRO -> <i>I</i>	2	2	1
V -> <i>hate</i>	2	4	.5
V -> <i>annoying</i>	1	4	.25
ADJ -> <i>annoying</i>	1	1	1
N -> <i>neighbors</i>	1	4	.25

Figure 5: The MLE Estimates from Trees in Figs 1 and 3

Given this model, we can now answer which interpretation in Figure 3 is more likely. The tree on the left (ignoring for the moment the words) involves rules S->NP VP, NP->PRO, VP->V NP, NP->ADJ N, PRO-> *I*, V->*hate*, ADJ->*annoying* and N->*neighbors*, and thus has a probability of  $1 * .29 * .75 * .15 * 1 * .5 * 1 * .25 = .004$ , whereas the tree on the right involves rules S->NP VP, NP->PRO, VP-> V VP, VP-> V NP, NP-> N, PRO-> *I*, V->*hate*, ADJ->*annoying* and N->*neighbors*, and thus has a probability of  $1 * .29 * .25 * .75 * .15 * 1 * .5 * 1 * .25 = .001$ . Thus the tree on the left appears more likely. Note that one reason that there is a large difference in probabilities is because the tree on the right involved one more rule. In general, PCFG will tend to prefer interpretations that involve the fewest number of rules.

Note that the probabilities of the lexical rules would be the same as the tag output probabilities we looked at when doing part of speech tagging, i.e,

$$P(\text{ADJ} \rightarrow \text{Annoying}) = P(\text{Annoying} \mid \text{ADJ})$$

## **Parsing Probabilistic Context Free Grammars**

To find the most likely parse tree, it might seem that we need to search an exponential number of possible trees. But using the probability model described above with its strong independence assumptions, we can use dynamic programming techniques to incrementally build up all possible constituents that cover one word, then two, and so on, over all words in the sentence. Since the probability of a tree is independent of how its subtrees were built, we can collapse the search at each point by just remembering the maximum probability parse. This algorithm looks similar to the min edit distance and Viterbi algorithms we developed earlier.

NP <sub>1</sub> .29		ADJ <sub>3</sub> 1	NP <sub>4</sub> .0375
PRO <sub>1</sub> 1	V <sub>2</sub> .5	V <sub>3</sub> .25	N <sub>4</sub> .25
I	hate	annoying	neighbors

Figure 6: The best probabilities for all constituents of length 1

Let's assign each word a number. So for the sentence *I hate annoying neighbors*, the word *I* would be 1, *hate* would be 2, and so on. Also, to simplify matters, let's assume all grammar rule are of only two possible forms, namely

X -> Y Z

X -> w, where w is a word

(i.e., there are 1 or 2 categories on the right hand side). This is called **Chomsky Normal Form**, and it can be shown that all context free grammars have an equivalent grammar in this form. The parsing algorithm is called the **probabilistic CKY algorithm**. It first builds all possible hypotheses of length one by considering all rules that could produce each of the words. Figure 6 shows all non-zero probabilities of the words, where we assume that *I* can be a pronoun (.9) or a noun (.1 - the letter "i"), *hate* is a verb, *annoying* is a verb (.7) or ADJ (.3), and *neighbors* is a noun. We first add all the lexical probabilities, and then use the rules to build any other constituents of length 1 as shown in Figure 6. For example, at position 1 we have an NP constructed from the rule NP -> PRO (prob .29) and constituent PRO<sub>1</sub> (prob .1), yielding NP<sub>1</sub> with prob .29. Likewise at position 4, we have an N with probability .25 (N<sub>4</sub>) which with rule NP->N (prob. .15) which produces an NP with probability .0375.

The next iteration constructs finds the maximum probability for all possible constituents of length two. In essence, we iterate though every non-terminal and every pair of points to find the maximum probability combination that produces that non-terminal (if there is any at all). Consider this for NP, where NP<sub>i,j</sub> will be an NP from position i to position j:

$P(NP_{1,2}) = 0$  (no rule that produces an NP has a non-zero LHS)

$P(NP_{2,3}) = 0$ , for same reasons

$P(NP_{3,4}) = P(NP \rightarrow ADJ N) * P(ADJ_3) * P(N_4) = .15 * 1 * .25 = .0375$ .

The only other non-zero constituent of length 2 in this simple example is VP<sub>3,4</sub>, which has probability  $P(VP \rightarrow V NP) * P(V_3) * P(NP_4) = .75 * .25 * .0375 = .007$ . The results after this iteration are shown as Figure 7.

The next iteration is building constituents of length 3. Here we have the first case we have

		VP <sub>3,4</sub> .007	
		NP <sub>3,4</sub> .0375	
NP <sub>1</sub> .29		ADJ <sub>3</sub> 1	NP <sub>4</sub> .0375
PRO <sub>1</sub> 1	V <sub>2</sub> .5	V <sub>3</sub> .25	N <sub>4</sub> .25
I	hate	annoying	neighbors

Figure 7: The best probabilities for all constituents of length 1 and 2

competing alternatives for the same constituent. In particular,  $VP_{2,4}$  has two possible derivations, using rule  $VP \rightarrow V NP$  or using rule  $VP \rightarrow V VP$ . Since we are interested only in finding the maximum parse tree here, we want whichever one gives the maximum probability.

$$\begin{aligned} VP \rightarrow V NP: & P(VP \rightarrow V NP) * P(V_2) * P(NP_{3,4}) = .75 * .5 * .0375 = .014 \\ VP \rightarrow V VP: & P(VP \rightarrow V VP) * P(V_2) * P(VP_{3,4}) = .25 * .5 * .007 = .0008 \end{aligned}$$

Thus we add  $VP_{3,4}$  with probability .014. Note that to recover the parse tree at the end, we would also have to record what rule and subconstituents were used to produce the maximum probability interpretation. As we move to the final iteration, note that there is only one possible combination to produce  $S_{1,4}$ , namely combining  $NP_1$  with  $VP_{2,4}$ . Because we dropped the other interpretation of the VP, that parse tree is not considered at the next level up. It is this property that allows us to search efficiently for the highest probability parse tree. There is only one possibility for a constituent of length four, combining  $NP_1$ ,  $VP_{2,4}$  and rule  $S \rightarrow NP VP$ , yielding the probability of  $S_{1,4}$  of .004 (as we got for this tree before).

Note that the CKY algorithm computed the highest probability tree for each constituent type over each possible word sequence, similar to the Viterbi algorithm finding the probability of the best path to each node and each time. A simple variant of the CKY algorithm could add all the probabilities of the trees together and produce an algorithm analogous to computing the forward probabilities in HMMs. This value is called the **inside probability** and is defined as follows, where  $w_{i,j}$  is the sequence of words from  $i$  to  $j$ ,  $N$  is a constituent name and  $N_{i,j}$  indicates that this constituent generates the input from positions  $i$  to  $j$ .

$$Inside_G(N, i, j) = P_G(w_{i,j} | N_{i,j})$$

i.e., the probability that the grammar  $G$  generates  $w_{i,j}$  given we know that constituent  $N$  is the root of the tree covering position  $i$  through  $j$ .

Another probability that is often used in algorithms involving PCFGs is the **outside probability**. This is the probability that a constituent appears between positions  $i$  and  $j$  given the context of the words on either side of the constituent (i.e., from 1 to  $i$  and  $j$  to the end). In other words,

$$outside_G(N, i, j) = P_G(w_{1,i-1}, N_{i,j}, w_{j+1,n})$$

If we put these two together, we get the probability that a constituent  $N$  generate the words between position  $i$  and  $j$  for a given input  $w_{1,n}$ .

$$P_G(N_{i,j} | w_{1,n}) = inside_k(N, i, j) * outside_k(N, i, j)$$

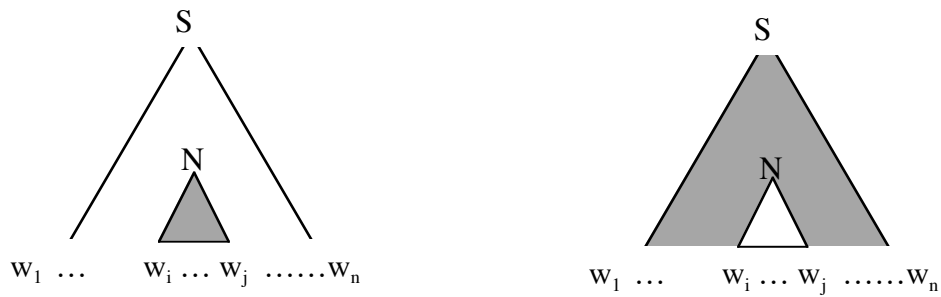


Figure 8: Inside and Outside Probabilities

### ***Evaluating Parser Accuracy***

A quick look at the literature will reveal that statistical parsers appear to be performing very well on quite challenging texts. For example, a common test corpus is text from the Wall Street Journal, which contains many very long sentences with quite complex structure. The best parsers report that that attain about a 90% accuracy in identifying the correct constituents. Seeing this, one might conclude that parsing is essentially solved. But this is not true. Once we look at the details in these claims, we see there is much that remains to be done. First, a 90% constituent accuracy sounds good, but what does this translate into as far as how many sentences are parsed accurately. Lets say we have a sentence that involves 20 constituents. Then the probability of getting a sentence completely correct is  $.9^{20} = .28$ . So the parser would only be getting 28% of the parses completely correct. Second, the results also depend on the constraints on the grammar itself. For instance, its easy to write a context free grammar that generates all sentences with two rules:

$S \rightarrow w$   
 $S \rightarrow S S$

i.e., a sentence is a sequence of words. Clearly getting the correct parse for this grammar would be trivial. Thus to evaluate the results, we need to consider whether the grammars are producing structures that will be useful for subsequent analysis. We should require, for instance, that the grammar requires that we identify all the major constituents in the sentences. In most grammatical models, the major constituents involves noun phrases, verb phrases and sentences, and adjective and adverbial phrases. Typically, the grammars being evaluated do meet this requirements. But for certain constructs that are important for semantic analyses (like whether a phrase is an argument to a verb or a modifier of the verb), the current grammars collapse these distinctions. So if we want a grammar for semantic analysis, the results will look even worse. The lesson here is not to accept accuracy numbers without a careful consideration of what assumptions are being made and how the results might be used.