Probabilistic Context-Free Grammars and beyond

Mark Johnson

Microsoft Research / Brown University

July 2007

Outline

What is computational linguistics?

Computational linguistics studies the *computational processes* involved in language

- Scientific questions:
  - How is language *comprehended* and *produced*?
  - How is language *acquired* (learned)?

  These are characterized in terms of *information processing*
- Technological applications:
  - Machine translation
  - Information extraction and question answering
  - Automatic summarization
  - Automatic speech recognition (?)

Why grammars?

Grammars specify a wide range of *sets of structured objects*

- especially useful for describing human languages
- applications in vision, computational biology, etc

There is a *hierarchy* of kinds of grammars

- if a language can be specified by a grammar low in the hierarchy, then it can be specified by a grammar higher in the hierarchy
- the location of a grammar in this hierarchy determines its computational properties

There are generic algorithms for *computing with* and *estimating* (learning) each kind of grammar

- no need to devise new models and corresponding algorithms for each new set of structures

Outline

Preview of (P)CFG material

- Why are context-free grammars called “context-free”?
- Context-free grammars (CFG) derivations and parse trees
- Probabilistic CFGs (PCFGs) define probability distributions over derivations/trees
- The number of derivations often grows exponentially with sentence length
- Even so, we can compute the sum/max of probabilities of all trees in cubic time
- It’s easy to estimate PCFGs from a treebank (a sequence of trees)
- The EM algorithm can estimate PCFGs from a corpus of strings

Formal languages

T is a finite set of *terminal symbols*, the vocabulary of the language

- E.g., $T = \{\mathsf{likes}, \mathsf{Sam}, \mathsf{Sasha}, \mathsf{thinks}\}$

A *string* is a *finite sequence* of elements of T

- E.g., Sam thinks Sam likes Sasha

$T^\star$ is the set of all strings (including the *empty string* ϵ)

T^+ is the set of all non-empty strings

A (formal) *language* is a set of strings (a subset of $T^\star$)

- E.g., $L = \{\mathsf{Sam}, \mathsf{Sam\ thinks}, \mathsf{Sasha\ thinks}, \ldots\}$

A *probabilistic language* is a probability distribution over a language

Rewrite grammars

A rewrite grammar $G = (T, N, S, R)$ consists of

- T, a finite set of *terminal symbols*
- N, a finite set of *nonterminal symbols* disjoint from T
- $S \in N$ is the *start symbol*, and
- R is a finite subset of $N^+ \times (N \cup T)^\star$

The members of R are called *rules* or *productions*, and usually written $\alpha \to \beta$, where $\alpha \in N^+$ and $\beta \in (N \cup T)^\star$

A rewrite grammar defines the *rewrites relation* $\Rightarrow$, where $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ iff $\alpha \to \beta \in R$ and $\gamma, \delta \in (N \cup T)^\star$.

A *derivation* of a string $w \in T^\star$ is a finite sequence of rewritings $S \Rightarrow \ldots \Rightarrow w$.

$\Rightarrow^\star$ is the reflexive transitive closure of $\Rightarrow$

The language *generated* by G is $\{w : S \Rightarrow^\star w, w \in T^\star\}$

Example of a rewriting grammar

$G_1 = (T_1, N_1, \mathsf{S}, R_1)$, where

$$T_1 = \{\mathsf{Al}, \mathsf{George}, \mathsf{snores}\},$$
$$N_1 = \{\mathsf{S}, \mathsf{NP}, \mathsf{VP}\},$$
$$R_1 = \{\mathsf{S} \rightarrow \mathsf{NP\ VP}, \mathsf{NP} \rightarrow \mathsf{Al}, \mathsf{NP} \rightarrow \mathsf{George}, \mathsf{VP} \rightarrow \mathsf{snores}\}.$$

Sample derivations:

$$\mathsf{S} \Rightarrow \mathsf{NP\ VP} \Rightarrow \mathsf{Al\ VP} \Rightarrow \mathsf{Al\ snores}$$
$$\mathsf{S} \Rightarrow \mathsf{NP\ VP} \Rightarrow \mathsf{George\ VP} \Rightarrow \mathsf{George\ snores}$$

The Chomsky Hierarchy

Grammars classified by the shape of their productions $\alpha \to \beta$.

Context-sensitive: $|\alpha| \leq |\beta|$
Context-free: $|\alpha| = 1$
Right-linear: $|\alpha| = 1$ and $\beta \in T^{\star}(N \cup \epsilon)$.

The classes of languages generated by these classes of grammars form a strict hierarchy (ignoring ϵ).

Language class	*Recognition complexity*
Unrestricted	undecidable
Context-sensitive	exponential time
Context-free	polynomial time
Linear	linear time

Right linear grammars define *finite state languages*, and probabilistic right linear grammars define the same distributions as finite state Hidden Markov Models.

Context-sensitivity in human languages

Some human languages are not context-free (Shieber 1984, Culy 1984).

Context-sensitive grammars don't seem useful for describing human languages.

Trees are intuitive descriptions of linguistic structure and are *normal forms* for context-free grammar derivations.

There is an infinite hierarchy of language families (and grammars) between context-free and context-sensitive.

Mildly context-sensitive grammars, such as Tree Adjoining Grammars (Joshi) and Combinatory Categorial Grammar (Steedman) seem useful for natural languages.

Parse trees for context-free grammars

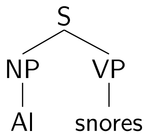
A *parse tree* generated by CFG $G = (T, N, S, R)$ is a finite ordered tree labeled with labels from $N \cup T$, where:

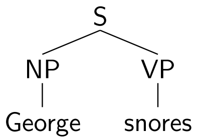
- the root node is labeled S
- for each node n labeled with a nonterminal $A \in N$ there is a rule $A \rightarrow \beta \in R$ and n's children are labeled β
- each node labeled with a terminal has no children

Ψ_G is the set of all parse trees generated by G.

$\Psi_G(w)$ is the subset of Ψ_G with yield $w \in T^\star$.

$$R_1 = \{\mathsf{S} \rightarrow \mathsf{NP\ VP}, \mathsf{NP} \rightarrow \mathsf{Al}, \mathsf{NP} \rightarrow \mathsf{George}, \mathsf{VP} \rightarrow \mathsf{snores}\}$$





Example of a CF derivation and parse tree

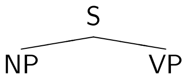
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$

S

S

Example of a CF derivation and parse tree

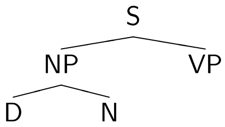
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$



$$\begin{array}{c} \mathsf{S} \\ \Rightarrow \quad \mathsf{NP\ VP} \end{array}$$

Example of a CF derivation and parse tree

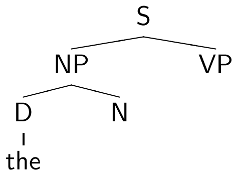
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \mathsf{S} \\ \Rightarrow & \mathsf{NP\ VP} \\ \Rightarrow & \mathsf{D\ N\ VP} \end{array}$$

Example of a CF derivation and parse tree

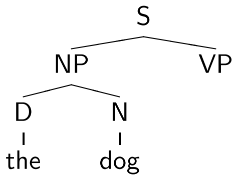
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$



$$\begin{array}{cc} & \mathsf{S} \\ \Rightarrow & \mathsf{NP\ VP} \\ \Rightarrow & \mathsf{D\ N\ VP} \\ \Rightarrow & \mathsf{the\ N\ VP} \end{array}$$

Example of a CF derivation and parse tree

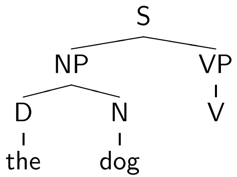
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \mathsf{S} \\ \Rightarrow & \mathsf{NP\ VP} \\ \Rightarrow & \mathsf{D\ N\ VP} \\ \Rightarrow & \mathsf{the\ N\ VP} \\ \Rightarrow & \mathsf{the\ dog\ VP} \end{array}$$

Example of a CF derivation and parse tree

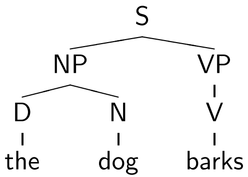
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \mathsf{S} \\ \Rightarrow & \mathsf{NP\ VP} \\ \Rightarrow & \mathsf{D\ N\ VP} \\ \Rightarrow & \mathsf{the\ N\ VP} \\ \Rightarrow & \mathsf{the\ dog\ VP} \\ \Rightarrow & \mathsf{the\ dog\ V} \end{array}$$

Example of a CF derivation and parse tree

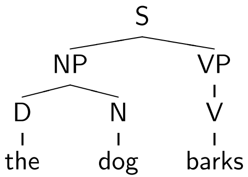
$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \mathsf{S} \\ \Rightarrow & \mathsf{NP\ VP} \\ \Rightarrow & \mathsf{D\ N\ VP} \\ \Rightarrow & \mathsf{the\ N\ VP} \\ \Rightarrow & \mathsf{the\ dog\ VP} \\ \Rightarrow & \mathsf{the\ dog\ V} \\ \Rightarrow & \mathsf{the\ dog\ barks} \end{array}$$

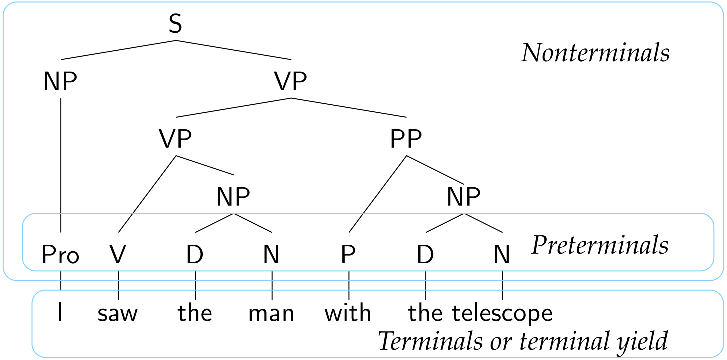
Example of a CF derivation and parse tree

$$R_2 = \left\{ \begin{array}{ccc} \mathsf{S} \rightarrow \mathsf{NP\ VP} & \mathsf{NP} \rightarrow \mathsf{D\ N} & \mathsf{VP} \rightarrow \mathsf{V} \\ \mathsf{D} \rightarrow \mathsf{the} & \mathsf{N} \rightarrow \mathsf{dog} & \mathsf{V} \rightarrow \mathsf{barks} \end{array} \right\}$$

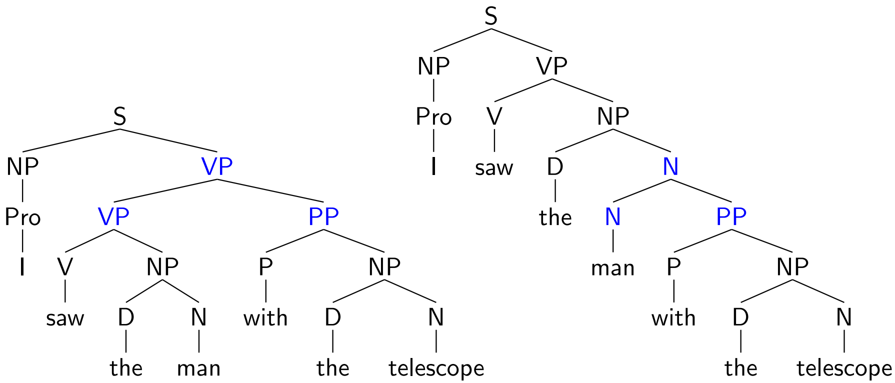


$$\begin{array}{rc} & \mathsf{S} \\ \Rightarrow & \mathsf{NP\ VP} \\ \Rightarrow & \mathsf{D\ N\ VP} \\ \Rightarrow & \mathsf{the\ N\ VP} \\ \Rightarrow & \mathsf{the\ dog\ VP} \\ \Rightarrow & \mathsf{the\ dog\ V} \\ \Rightarrow & \mathsf{the\ dog\ barks} \end{array}$$

Trees can depict constituency



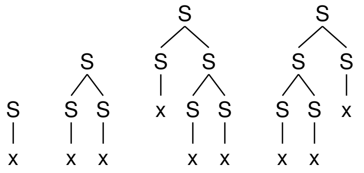
CFGs can describe structural ambiguity

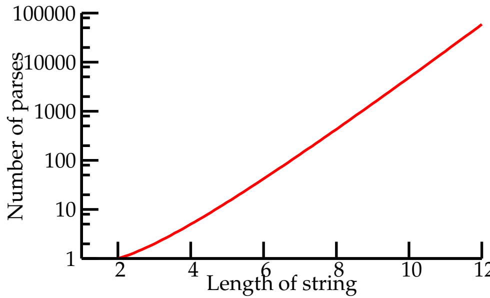


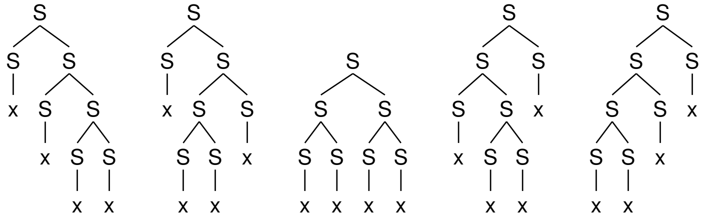
$$R_2 = \{\mathsf{VP} \rightarrow \mathsf{V\ NP}, \mathsf{VP} \rightarrow \mathsf{VP\ PP}, \mathsf{NP} \rightarrow \mathsf{D\ N}, \mathsf{N} \rightarrow \mathsf{N\ PP}, \ldots\}$$

Ambiguities usually grow exponentially with length

$$R = \{\mathsf{S} \rightarrow \mathsf{S}\ \mathsf{S}, \mathsf{S} \rightarrow \mathsf{x}\}$$





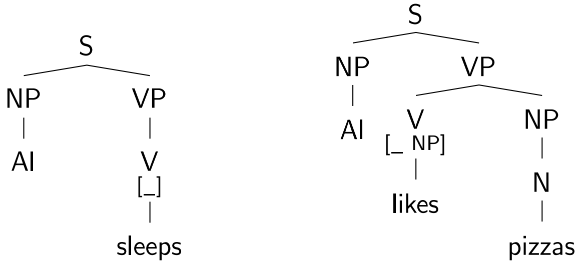


CFG account of subcategorization

Nouns and verbs differ in the number of *complements* they appear with.

We can use a CFG to describe this by splitting or *subcategorizing* the basic categories.

$$R_4 = \{\mathsf{VP} \to \underset{[\_]}{\mathsf{V}}, \mathsf{VP} \to \underset{[\_\,\mathsf{NP}]}{\mathsf{V}}\ \mathsf{NP}, \underset{[\_]}{\mathsf{V}} \to \mathsf{sleeps}, \underset{[\_\,\mathsf{NP}]}{\mathsf{V}} \to \mathsf{likes}, \ldots\}$$

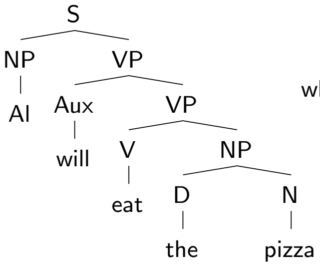


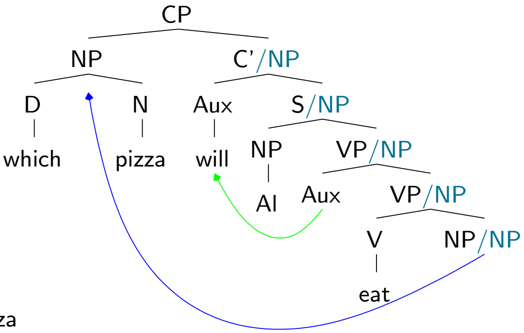
Nonlocal “movement” constructions

“Movement” constructions involve a phrase appearing far from its normal location.

Linguists believed CFGs could not generate them, and posited “movement transformations” to produce them (Chomsky 1957)

But CFGs can generate them via “feature passing” using a conspiracy of rules (Gazdar 1984)





Outline

Probabilistic grammars

A probabilistic grammar G defines a probability distribution $\mathrm{P}_G(\psi)$ over the parse trees Ψ generated by G, and hence over strings generated by G.

$$\mathrm{P}_G(w) \;=\; \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$$

Standard (non-stochastic) grammars distinguish *grammatical* from *ungrammatical* strings (only the grammatical strings receive parses).

Probabilistic grammars can assign non-zero probability to every string, and rely on the probability distribution to distinguish likely from unlikely strings.

Probabilistic context-free grammars

A *Probabilistic Context Free Grammar* (PCFG) consists of (T, N, S, R, p) where:

- (T, N, S, R) is a CFG with no useless productions or nonterminals, and
- p is a vector of *production probabilities*, i.e., a function $R \rightarrow [0, 1]$ that satisfies for each $A \in N$:

$$\sum_{A \rightarrow \beta \in R(A)} p(A \rightarrow \beta) \; = \; 1$$

  where $R(A) = \{A \rightarrow \alpha : A \rightarrow \alpha \in R\}$.

A production $A \rightarrow \alpha$ is *useless* iff there are no derivations of the form $S \Rightarrow^\star \gamma A \delta \Rightarrow \gamma \alpha \delta \Rightarrow^* w$ for any $\gamma, \delta \in (N \cup T)^\star$ and $w \in T^\star$.

Probability distribution defined by a PCFG

Intuitive interpretation:

- the probability of rewriting nonterminal A to α is $p(A \to \alpha)$
- the probability of a derivation is the product of probabilities of rules used in the derivation

For each production $A \to \alpha \in R$, let $f_{A\to\alpha}(\psi)$ be the number of times $A \to \alpha$ is used in ψ.

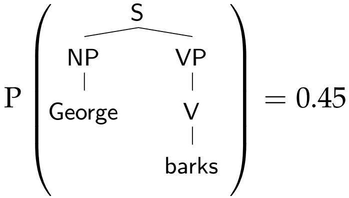
A PCFG G defines a probability distribution P_G on Ψ that is non-zero on $\Psi(G)$:

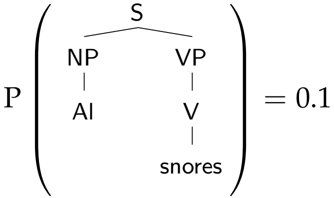
$$\mathrm{P}_G(\psi) \;=\; \prod_{r \in R} p(r)^{f_r(\psi)}$$

This distribution is properly normalized if p satisfies suitable constraints.

Example PCFG

1.0	S $\longrightarrow$ NP VP	1.0	VP $\longrightarrow$ V
0.75	NP $\longrightarrow$ George	0.25	NP $\longrightarrow$ Al
0.6	VP $\longrightarrow$ barks	0.4	VP $\longrightarrow$ snores





PCFGs as recursive mixtures

The distributions over strings induced by a PCFG in *Chomsky-normal form* (i.e., all productions are of the form $A \to B\,C$ or $A \to x$, where $A, B, C \in N$ and $x \in T$) is G_S where:

$$G_A = \sum_{A \to B\,C \in R_A} p A \to B\,C G_B \bullet G_C + \sum_{A \to w \in R_A} p A \to x \delta_x$$

$$(P \bullet Q)(z) = \sum_{xy=z} P(x) Q(y)$$

$$\delta_x(w) = 1 \text{ if } w = x \text{ and } 0 \text{ otherwise}$$

In fact, $G_A(w) = \mathrm{P}(A \Rightarrow^\star w | \theta)$, the sum of the probability of all trees with root node A and yield w

Outline

Hidden Markov Models and Finite State Automata

- Finite State Automata (FSA) are probably the simplest devices that generate an infinite number of strings
- They are conceptually and computationally simpler than CFGs, but can only express a subset of CF languages
- They are expressive enough for many useful tasks:
  - Speech recognition
  - Phonology and morphology
  - Lexical processing
- Very large FSA can be built and used very efficiently
- If the states are not visible, then Finite State Automata define Hidden Markov Models

Informal description of Finite State Automata

FSA generate arbitrarily long strings one symbol at a time.

At each step the FSA is in one of *a finite number of states*.

A FSA generates strings as follows:

1. Initialize the machine's state s to the *start state*
2. Loop: Based on the current state s, either
    2.1 stop, or
    2.2 emit a terminal x move to state s'

The language the FSA generates is the set of all sequences of terminals it can emit.

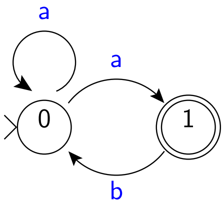
In a *probabilistic FSA*, these actions are directed by probability distributions.

If the states are not visible, this is a *Hidden Markov Model*

(Mealy) finite-state automata

A *(Mealy) automaton* $M = (T, N, S, F, M)$ consists of:

- T, a set of *terminals*, ($T = \{\mathsf{a}, \mathsf{b}\}$)
- N, a finite set of *states*, ($N = \{0, 1\}$)
- $S \in N$, the *start state*, ($S = 0$)
- $F \subseteq N$, the set of *final states* ($F = \{1\}$) and
- $M \subseteq N \times T \times N$, the *state transition relation*. ($M = \{(0, \mathsf{a}, 0), (0, \mathsf{a}, 1), (1, \mathsf{b}, 0)\}$)


$$\begin{tikzcd} \arrow[r] & 0 \arrow[loop above, "a"] \arrow[r, bend left, "a"] & 1 \arrow[l, bend left, "b"] \end{tikzcd}$$

A *accepting derivation* of a string $w_1 \ldots w_n \in T^\star$ is a sequence of states $s_0 \ldots s_n \in NS^\star$ where:

- $s_0 = S$ is the start state
- $s_n \in F$, and
- for each $i = 1 \ldots n$, $(s_{i-1}, v_i, s_i) \in M$.

00101 is an accepting derivation of aaba.

Probabilistic Mealy automata

A *probabilistic Mealy automaton* $M = (T, N, S, p_f, p_m)$ consists of:

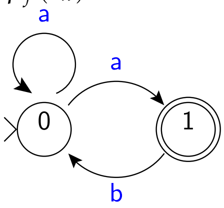
- terminals T, states N and start state $S \in N$ as before,
- $p_f(s)$, the probability of *halting at state* $s \in N$, and
- $p_m(v, s'|s)$, the probability of *moving from* $s \in N$ *to* $s' \in N$ *and emitting a* $v \in T$.

where $p_f(s) + \sum_{v \in T, s' \in N} p_m(v, s'|s) = 1$ for all $s \in \mathcal{S}$ (halt or move on)

The probability of a derivation with states $s_0 \dots s_n$ and outputs $v_1 \dots v_n$ is:

$$\mathrm{P}_M(s_0 \dots s_n; v_1 \dots v_n) = \left(\prod_{i=1}^{n} p_m(v_i, s_i | s_{i-1}) \right) p_f(s_n)$$

Example: $p_f(0) = 0, p_f(1) = 0.1$,
$p_m(\mathsf{a}, 0|0) = 0.2, p_m(\mathsf{a}, 1|0) = 0.8, p_m(\mathsf{b}, 0|1) = 0.9$
$P_M(00101, \mathsf{aaba}) = 0.2 \times 0.8 \times 0.9 \times 0.8 \times 0.1$

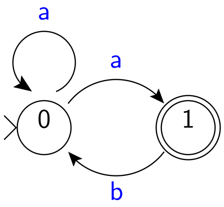


Probabilistic FSA as PCFGs

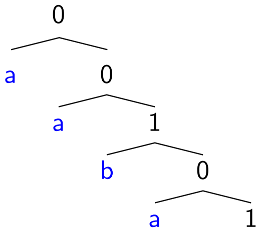
Given a Mealy PFSA $M = (T, N, S, p_f, p_m)$, let G_M have the same terminals, states and start state as M, and have productions

- $s \to \epsilon$ with probability $p_f(s)$ for all $s \in N$
- $s \to v\, s'$ with probability $p_m(v, s'|s)$ for all $s, s' \in N$ and $v \in T$

$p(0 \to \mathsf{a}\ 0) = 0.2, p(0 \to \mathsf{a}\ 1) = 0.8,$
$p(1 \to \epsilon) = 0.1, p(1 \to \mathsf{b}\ 0) = 0.9$



Mealy FSA



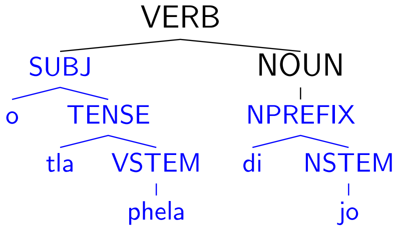
PCFG parse of aaba

PCFGs can express "multilayer HMMs"

Many applications involve hierarchies of FSA.

PCFGs provide a systematic way of describing such systems.

Sesotho is an *agglutinative language* in which each word contains several morphemes. Each part of speech (noun, verb) is generated by an FSA. Another FSA generates the possible sequences of parts of speech.



Outline

Things we want to compute with PCFGs

Given a PCFG G and a string $w \in T^\star$,

- (parsing): the most likely tree for w,

$$\operatorname{argmax}_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$$

- (language modeling): the probability of w,

$$\mathrm{P}_G(w) \;=\; \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi)$$

Learning rule probabilities from data:

- (maximum likelihood estimation from visible data): given a corpus of trees $\boldsymbol{d} = (\psi_1, \ldots, \psi_n)$, which rule probabilities p makes $\boldsymbol{d}$ as likely as possible?
- (maximum likelihood estimation from hidden data): given a corpus of strings $\boldsymbol{w} = (w_1, \ldots, w_n)$, which rule probabilities p makes $\boldsymbol{w}$ as likely as possible?

Parsing and language modeling

The probability $\mathrm{P}_G(\psi)$ of a tree $\psi \in \Psi_G(w)$ is:

$$\mathrm{P}_G(\psi) \;=\; \prod_{r \in R} p(r)^{f_r(\psi)}$$

Suppose the set of parse trees $\Psi_G(w)$ is finite, and we can enumerate it.

Naive parsing/language modeling algorithms for PCFG G and string $w \in T^\star$:

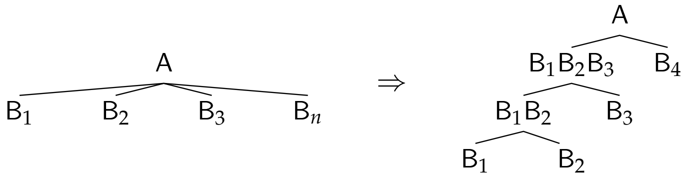
1. Enumerate the set of parse trees $\Psi_G(w)$
2. Compute the probability of each $\psi \in \Psi_G(w)$
3. Argmax/sum as appropriate

Chomsky normal form

A CFG is in *Chomsky Normal Form* (CNF) iff all productions are of the form $A \rightarrow B\ C$ or $A \rightarrow x$, where $A, B, C \in N$ and $x \in T$.

PCFGs *without epsilon productions* $A \rightarrow \epsilon$ can always be put into CNF.

Key step: *binarize* productions with more than two children by introducing new nonterminals



Substrings and string positions

Let $w = w_1 w_2 \ldots w_n$ be a string of length n

A *string position* for w is an integer $i \in 0, \ldots, n$ (informally, it identifies the position between words w_{i-1} and w_i)

●	the	●	dog	●	chases	●	cats	●
0		1		2		3		4

A *substring* of w can be specified by beginning and ending string positions

$w_{i,j}$ is the substring starting at word $i + 1$ and ending at word j.

$w_{0,4}$ = the dog chases cats
$w_{1,2}$ = dog
$w_{2,4}$ = chases cats

Language modeling using dynamic programming

- *Goal:* To compute $\mathrm{P}_G(w) = \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi) = \mathrm{P}_G(S \Rightarrow^* w)$
- *Data structure:* A table called a *chart* recording $\mathrm{P}_G(A \Rightarrow^* w_{i,k})$ for all $A \in N$ and $0 \leq i < k \leq |w|$
- *Base case:* For all $i = 1, \ldots, n$ and $A \to w_i$, compute:

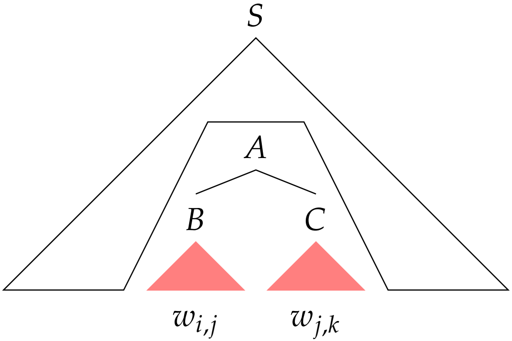
$$\mathrm{P}_G(A \Rightarrow^* w_{i-1,i}) \;=\; p(A \to w_i)$$

- *Recursion:* For all $k - i = 2, \ldots, n$ and $A \in N$, compute:

$$\begin{aligned} &\mathrm{P}_G(A \Rightarrow^* w_{i,k}) \\ &\quad = \sum_{j=i+1}^{k-1} \sum_{A \to B\,C \in R(A)} p(A \to B\,C)\mathrm{P}_G(B \Rightarrow^* w_{i,j})\mathrm{P}_G(C \Rightarrow^* w_{j,k}) \end{aligned}$$

Dynamic programming recursion

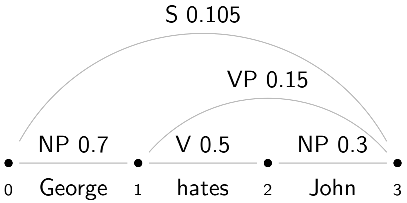
$$\mathrm{P}_G(A \Rightarrow^* w_{i,k}) = \sum_{j=i+1}^{k-1} \sum_{A \to B\,C \in R(A)} p(A \to B\,C)\mathrm{P}_G(B \Rightarrow^* w_{i,j})\mathrm{P}_G(C \Rightarrow^* w_{j,k})$$



$\mathrm{P}_G(A \Rightarrow^* w_{i,k})$ is called the *inside probability* of A spanning $w_{i,k}$.

Example PCFG string probability calculation

$$
\begin{array}{rcl}
w & = & \text{George hates John} \\
R & = & \left\{ \begin{array}{llllll}
1.0 & \text{S} \to \text{NP VP} & & 1.0 & \text{VP} \to \text{V NP} \\
0.7 & \text{NP} \to \text{George} & & 0.3 & \text{NP} \to \text{John} \\
0.5 & \text{V} \to \text{likes} & & 0.5 & \text{V} \to \text{hates}
\end{array} \right\}
\end{array}
$$

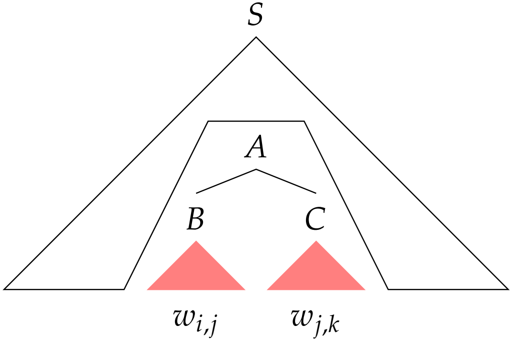


Right string position

Left string position	1	2	3
0	NP 0.7		S 0.105
1		V 0.5	VP 0.15
2			NP 0.3

Computational complexity of PCFG parsing

$$\begin{aligned}&\mathrm{P}_G(A \Rightarrow^* w_{i,k})\\ &\quad = \sum_{j=i+1}^{k-1} \sum_{A \to B\,C \in R(A)} p(A \to B\,C)\,\mathrm{P}_G(B \Rightarrow^* w_{i,j})\,\mathrm{P}_G(C \Rightarrow^* w_{j,k})\end{aligned}$$



For each production $r \in R$ and each i, k, we must sum over all intermediate positions $j \Rightarrow O(n^3|R|)$ time

Outline

Estimating (learning) PCFGs from data

Estimating productions and production probabilities from *visible data* (corpus of parse trees) is straight-forward:

- the productions are identified by the local trees in the data
- *Maximum likelihood principle:* select production probabilities in order to make corpus as likely as possible
- Bayesian estimators often produce more useful estimates

Estimating production probabilities from *hidden data* (corpus of terminal strings) is much more difficult:

- The *Expectation-Maximization* (EM) algorithm finds probabilities that *locally maximize* likelihood of corpus
- The *Inside-Outside* algorithm runs in time polynomial in length of corpus
- Bayesian estimators have recently been developed

Estimating the productions from hidden data is an open problem.

Estimating PCFGs from visible data

Data: A *treebank* of parse trees $\Psi = \psi_1, \ldots, \psi_n$.

$$L(p) = \prod_{i=1}^{n} \mathrm{P}_G(\psi_i) = \prod_{A \to \alpha \in R} p(A \to \alpha)^{f_{A\to\alpha}(\Psi)}$$

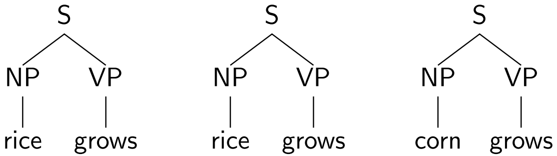
Introduce $|N|$ Lagrange multipliers $c_B, B \in N$ for the constraints $\sum_{B\to\beta\in R(B)} p(B \to \beta) = 1$:

$$\frac{\partial \left(L(p) - \sum_{B \in N} c_B \left(\sum_{B\to\beta\in R(B)} p(B\to\beta) - 1 \right) \right)}{\partial p(A \to \alpha)} = \frac{L(p) f_r(\Psi)}{p(A\to\alpha)} - c_A$$

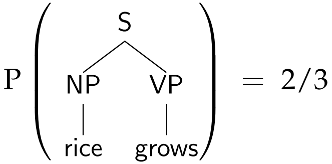
Setting this to 0, $\quad p(A \to \alpha) = \dfrac{f_{A\to\alpha}(\Psi)}{\sum_{A\to\alpha'\in R(A)} f_{A\to\alpha'}(\Psi)}$

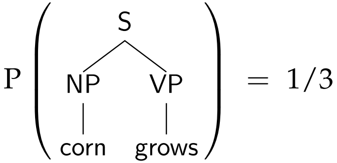
Visible PCFG estimation example

$\Psi =$



Rule	**Count**	**Rel Freq**
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	2/3
NP $\rightarrow$ corn	1	1/3
VP $\rightarrow$ grows	3	1





Estimating production probabilities from hidden data

Data: A corpus of sentences $\boldsymbol{w} = w_1, \ldots, w_n$.

$$L(\boldsymbol{w}) \;=\; \prod_{i=1}^{n} \mathrm{P}_G(w_i). \qquad \mathrm{P}_G(w) \;=\; \sum_{\psi \in \Psi_G(w)} \mathrm{P}_G(\psi).$$

$$\frac{\partial L(\boldsymbol{w})}{\partial p(A \to \alpha)} \;=\; \frac{L(\boldsymbol{w}) \sum_{i=1}^{n} \mathrm{E}_G(f_{A\to\alpha}|w_i)}{p(A \to \alpha)}$$

Setting this equal to the Lagrange multiplier c_A and imposing the constraint $\sum_{B\to\beta\in R(B)} p(B \to \beta) = 1$:

$$p(A \to \alpha) \;=\; \frac{\sum_{i=1}^{n} \mathrm{E}_G(f_{A\to\alpha}|w_i)}{\sum_{A\to\alpha'\in R(A)} \sum_{i=1}^{n} \mathrm{E}_G(f_{A\to\alpha'}|w_i)}$$

This is an iteration of the *expectation maximization* algorithm!

The EM algorithm for PCFGs

Input: a corpus of strings $\boldsymbol{w} = w_1, \ldots, w_n$

Guess initial production probabilities $p^{(0)}$

For $t = 1, 2, \ldots$ do:

1. Calculate *expected frequency* $\sum_{i=1}^{n} \mathrm{E}_{p^{(t-1)}}(f_{A\to\alpha}|w_i)$ of each production:

$$\mathrm{E}_p(f_{A\to\alpha}|w) \quad = \quad \sum_{\psi \in \Psi_G(w)} f_{A\to\alpha}(\psi) \mathrm{P}_p(\psi)$$

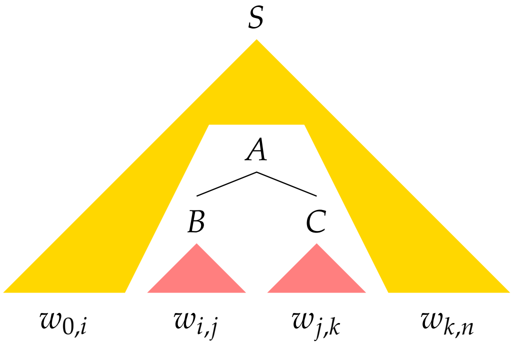
2. Set $p^{(t)}$ to the *relative expected frequency* of each production

$$p^{(t)}(A \to \alpha) \quad = \quad \frac{\sum_{i=1}^{n} \mathrm{E}_{p^{(t-1)}}(f_{A\to\alpha}|w_i)}{\sum_{A\to\alpha'} \sum_{i=1}^{n} \mathrm{E}_{p^{(t-1)}}(f_{A\to\alpha'}|w_i)}$$

It is as if $p^{(t)}$ were estimated from a visible corpus Ψ_G in which each tree ψ occurs $\sum_{i=1}^{n} \mathrm{P}_{p^{(t-1)}}(\psi|w_i)$ times.

Dynamic programming for $E_p(f_{A \to B\,C} | w)$

$$E_p(f_{A \to B\,C} | w) = \frac{\sum_{0 \le i < j < k \le n} \mathrm{P}(S \Rightarrow^* w_{1,i}\, A\, w_{k,n})\, p(A \to B\,C)\, \mathrm{P}(B \Rightarrow^* w_{i,j})\, \mathrm{P}(C \Rightarrow^* w_{j,k})}{\mathrm{P}_G(w)}$$



Calculating "outside probabilities"

Construct a table of "outside probabilities"
$\mathrm{P}_G(S \Rightarrow^* w_{0,i}\, A\, w_{k,n})$ for all $0 \leq i < k \leq n$ and $A \in N$

Recursion from *larger to smaller* substrings in w.

Base case: $\mathrm{P}(S \Rightarrow^* w_{0,0}\, S\, w_{n,n}) = 1$

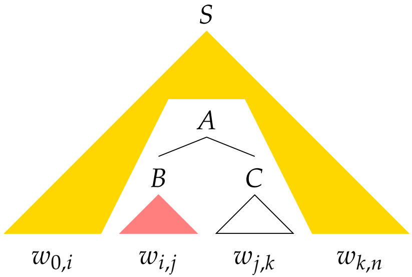
Recursion: $\mathrm{P}(S \Rightarrow^* w_{0,j}\, C\, w_{k,n}) =$

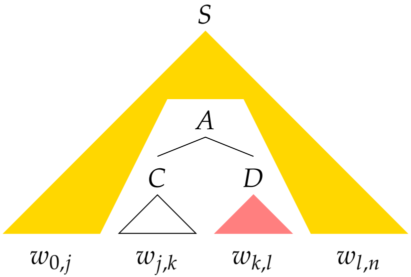
$$\sum_{i=0}^{j-1} \sum_{\substack{A,B \in N \\ A \to B\,C \in R}} \mathrm{P}(S \Rightarrow^* w_{0,i}\, A\, w_{k,n}) p(A \to B\, C) \mathrm{P}(B \Rightarrow^* w_{i,j})$$

$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in N \\ A \to C\,D \in R}} \mathrm{P}(S \Rightarrow^* w_{0,j}\, A\, w_{l,n}) p(A \to C\, D) \mathrm{P}(D \Rightarrow^* w_{k,l})$$

Recursion in $\mathrm{P}_G(S \Rightarrow^* w_{0,i}\, A\, w_{k,n})$

$$\mathrm{P}(S \Rightarrow^* w_{0,j}\, C\, w_{k,n}) = \sum_{i=0}^{j-1} \sum_{\substack{A,B \in N \\ A \to B\,C \in R}} \mathrm{P}(S \Rightarrow^* w_{0,i}\, A\, w_{k,n})\, p(A \to B\,C)\, \mathrm{P}(B \Rightarrow^* w_{i,j}) + \sum_{l=k+1}^{n} \sum_{\substack{A,D \in N \\ A \to C\,D \in R}} \mathrm{P}(S \Rightarrow^* w_{0,j}\, A\, w_{l,n})\, p(A \to C\,D)\, \mathrm{P}(D \Rightarrow^* w_{k,l})$$





Example: The EM algorithm with a toy PCFG

Initial rule probs

rule	prob
...	...
VP $\rightarrow$ V	0.2
VP $\rightarrow$ V NP	0.2
VP $\rightarrow$ NP V	0.2
VP $\rightarrow$ V NP NP	0.2
VP $\rightarrow$ NP NP V	0.2
...	...
Det $\rightarrow$ the	0.1
N $\rightarrow$ the	0.1
V $\rightarrow$ the	0.1

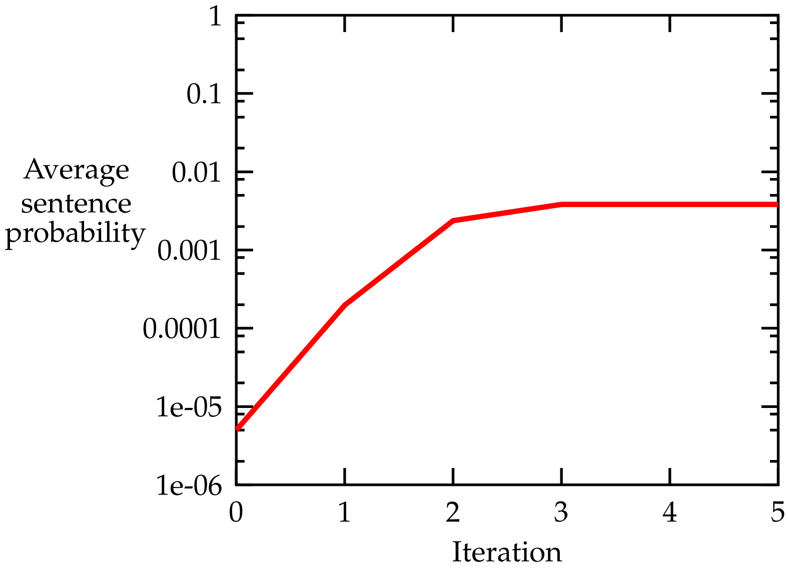
"English" input

the dog bites
the dog bites a man
a man gives the dog a bone
...

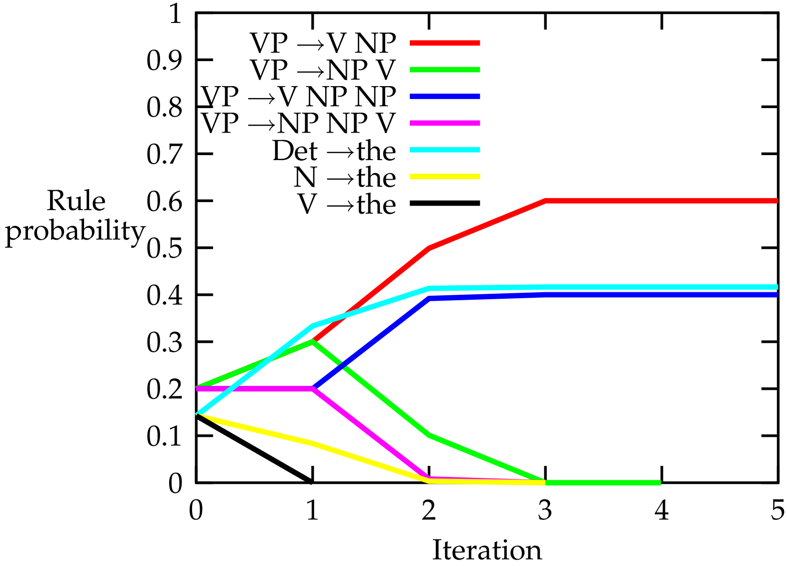
"pseudo-Japanese" input

the dog bites
the dog a man bites
a man the dog a bone gives
...

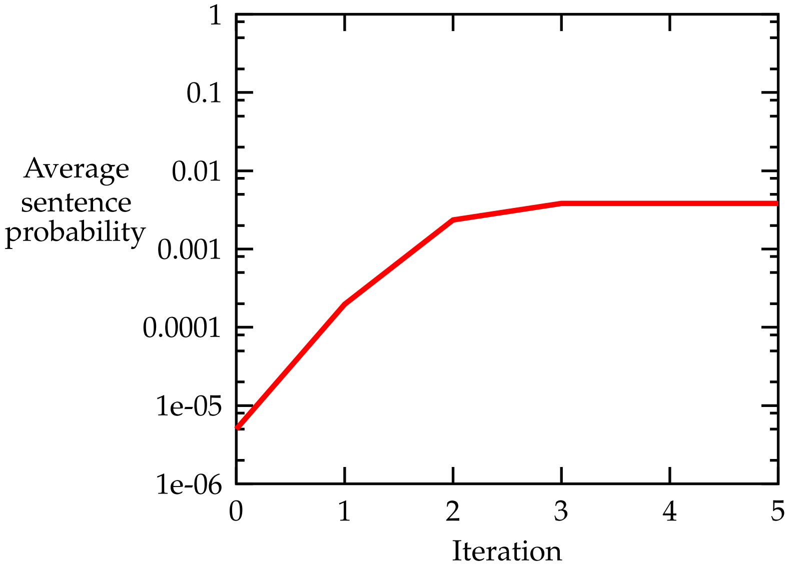
Probability of “English”



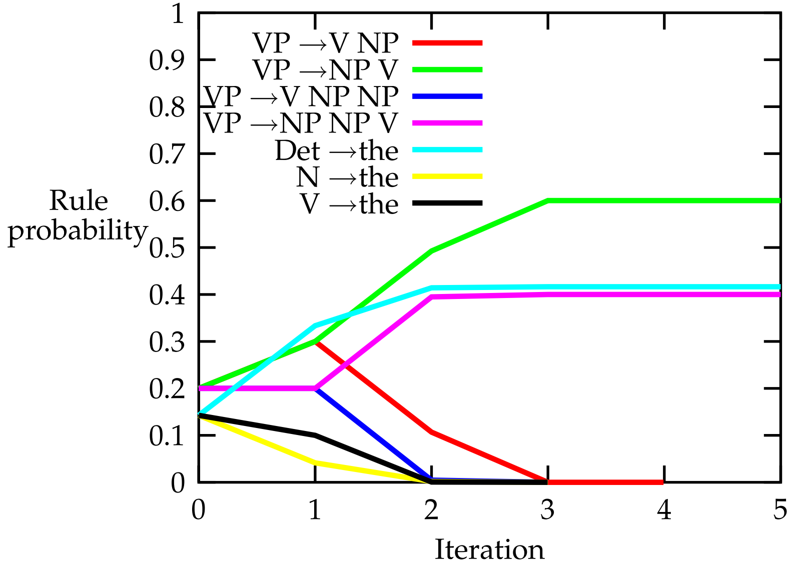
Rule probabilities from “English”



Probability of “Japanese”



Rule probabilities from “Japanese”

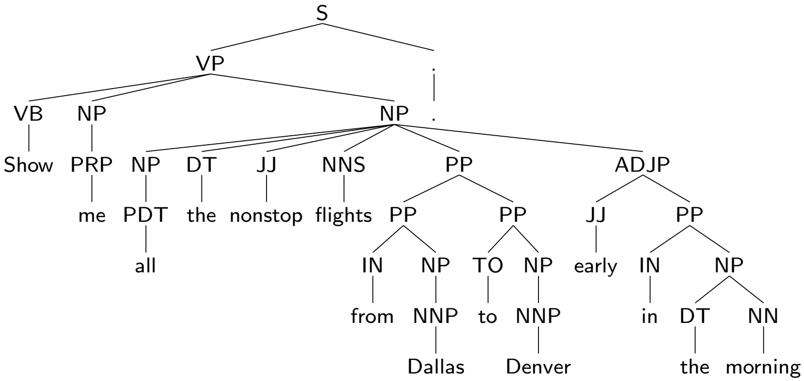


Learning in statistical paradigm

- The likelihood is a differentiable function of rule probabilities
  $\Rightarrow$ learning can involve small, incremental updates
- Learning structure (rules) is hard, but …
- Parameter estimation can approximate rule learning
  - start with “superset” grammar
  - estimate rule probabilities
  - discard low probability rules
- Non-parametric Bayesian estimators combine parameter and rule estimation

Applying EM to real data

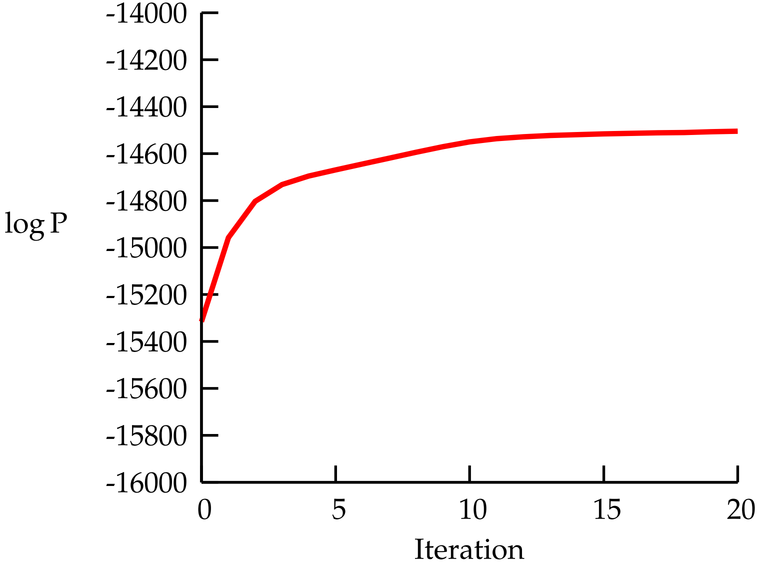
- ATIS treebank consists of 1,300 hand-constructed parse trees
- ignore the words (in this experiment)
- about 1,000 PCFG rules are needed to build these trees



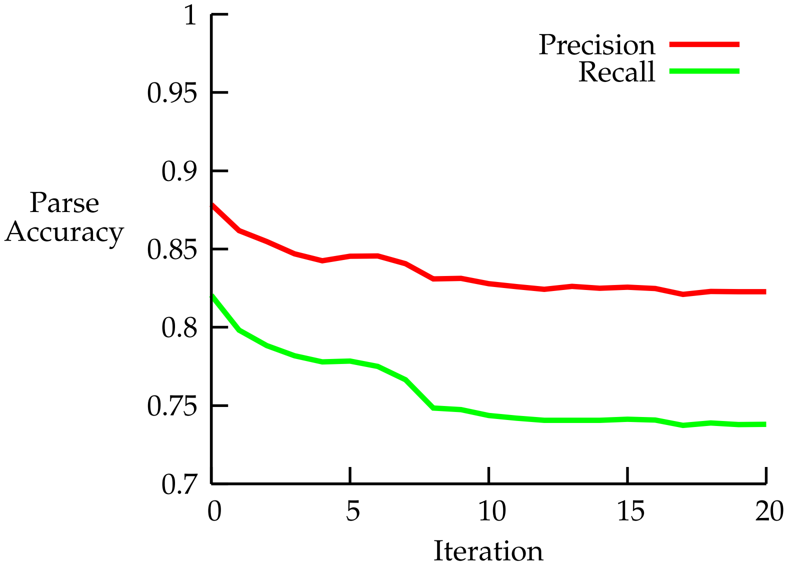
Experiments with EM on ATIS

1. Extract productions from trees and estimate probabilities probabilities from trees to produce PCFG.
2. Initialize EM with the treebank grammar and MLE probabilities
3. Apply EM (to strings alone) to re-estimate production probabilities.
4. At each iteration:
   - Measure the likelihood of the training data and the quality of the parses produced by each grammar.
   - Test on training data (so poor performance is not due to overlearning).

Likelihood of training strings



Quality of ML parses



Why does EM do so poorly?

- EM assigns trees to strings to maximize the marginal probability of the strings, but the trees weren't designed with that in mind
- We have an "intended interpretation" of categories like NP, VP, etc., which EM has no way of knowing
- Our *grammars are defective*
  - real language has dependencies that these PCFGs can't capture
- How can information about the *marginal distribution of strings* $\mathrm{P}(w)$ provide information about the *conditional distribution of parses given strings* $\mathrm{P}(\psi|w)$?
  - need additional *linking assumptions* about the relationship between parses and strings
- …but no one really knows.

Outline

Unsupervised inference for PCFGs

- Given a CFG G and corpus of strings $\mathbf{w}$, infer:
  - rule probabilities θ
  - trees $\mathbf{t}$ for $\mathbf{w}$
- Maximum likelihood, e.g. Inside-Outside/EM (a *point estimate*)

$$\begin{aligned} \hat{\theta} &= \operatorname{argmax}_\theta \mathrm{P}(\mathbf{w}|\theta) && \text{(EM)} \\ \hat{\mathbf{t}} &= \operatorname{argmax}_\mathbf{t} \mathrm{P}(\mathbf{t}|\mathbf{w},\hat{\theta}) && \text{(Viterbi)} \end{aligned}$$

- Bayesian inference incorporates *prior* $\mathrm{P}(\theta)$ and infers a *posterior distribution*

$$\begin{aligned} \underbrace{\mathrm{P}(\theta|\mathbf{w})}_{\text{Posterior}} &\propto \underbrace{\mathrm{P}(\mathbf{w}|\theta)}_{\text{Likelihood}} \; \underbrace{\mathrm{P}(\theta)}_{\text{Prior}} \\ \mathrm{P}(\mathbf{t}|\mathbf{w}) &\propto \int_\Delta \mathrm{P}(\mathbf{w},\mathbf{t}|\theta)\,\mathrm{P}(\theta)\,d\theta \end{aligned}$$

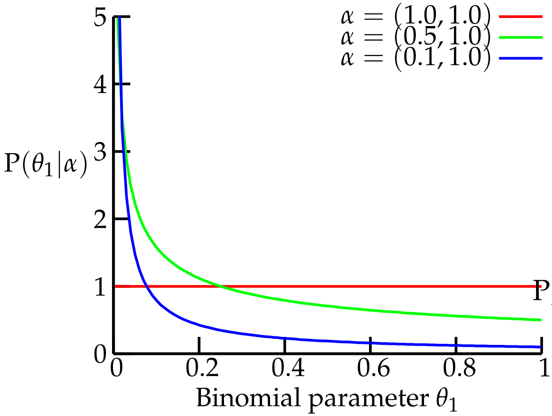
Bayesian priors

$$\underbrace{\mathrm{P}(\text{Hypothesis}|\text{Data})}_{\text{Posterior}} \propto \underbrace{\mathrm{P}(\text{Data}|\text{Hypothesis})}_{\text{Likelihood}} \underbrace{\mathrm{P}(\text{Hypothesis})}_{\text{Prior}}$$

- Hypothesis = rule probabilities θ, Data = strings $\mathbf{w}$
- Prior can incorporate linguistic insights ("universal grammar")
- Math/computation vastly simplified if *prior is conjugate to likelihood*
  - posterior belongs to the same model family as prior
- PCFGs are *products of multinomials*, one for each nonterminal A
  - model has a parameter $\theta_{A \to \beta}$ for each rule $A \to \beta \in R$

$\Rightarrow$ Conjugate prior is *product of Dirichlets*, one for each nonterminal A
  - prior has a hyper-parameter $\alpha_{A \to \beta}$ for each rule $A \to \beta \in R$

Dirichlet priors for multinomials



- Outcomes $1, \ldots, m$
- Multinomial
  $\mathrm{P}(X = i) = \theta_i$
  $\theta = (\theta_1, \ldots, \theta_m)$
- Dirichlet prior parameters
  $\alpha = (\alpha_1, \ldots, \alpha_m)$

$$\mathrm{P}_D(\theta | \alpha) = \frac{1}{Z(\alpha)} \prod_{i=1}^{m} \theta_i^{\alpha_i - 1}$$

$$Z(\alpha) = \frac{\prod_{i=1}^{m} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{m} \alpha_i)}$$

- As α_1 approaches 0, $\mathrm{P}(\theta_1 | \alpha)$ concentrates around 0
- PCFG prior is product of Dirichlets (one for each $A \in N$)
- Dirichlet for A in PCFG prior has hyper-parameter vector α_A
- Dirichlet prior can prefer *sparse grammars* in which $\theta_r = 0$

Dirichlet priors for PCFGs

- Let R_A be the rules expanding A in R, and θ_A, α_A be the subvectors of θ, α corresponding to R_A
- Conjugacy makes the posterior simple to compute given trees $\mathbf{t}$:

$$\begin{aligned}
\mathrm{P}_D(\theta|\alpha) &= \prod_{A \in N} \mathrm{P}_D(\theta_A|\alpha_A) \;\propto\; \prod_{r \in R} \theta^{\alpha_r} \\
\mathrm{P}(\theta|\mathbf{t},\alpha) &\propto \mathrm{P}(\mathbf{t}|\theta)\mathrm{P}_D(\theta|\alpha) \\
&\propto \left(\prod_{r \in R} \theta_r^{f_r(\mathbf{t})}\right) \left(\prod_{r \in R} \theta_r^{\alpha_r - 1}\right) \\
&= \prod_{r \in R} \theta_r^{f_r(\mathbf{t}) + \alpha_r - 1}, \quad \text{so} \\
\mathrm{P}(\theta|\mathbf{t},\alpha) &= \mathrm{P}_D(\theta|\mathbf{f}(\mathbf{t}) + \alpha)
\end{aligned}$$

- So when trees $\mathbf{t}$ are observed, posterior is product of Dirichlets
- But what if trees $\mathbf{t}$ are hidden, and only strings $\mathbf{w}$ are observed?

Algorithms for Bayesian inference

- Posterior is *computationally intractable*

$$\mathrm{P}(\mathbf{t}, \theta | \mathbf{w}) \quad \propto \quad \mathrm{P}(\mathbf{w}, \mathbf{t} | \theta)\, \mathrm{P}(\theta)$$

- Maximum A Posteriori (MAP) estimation finds the posterior mode

$$\theta^\star \quad = \quad \mathrm{argmax}_\theta \mathrm{P}(\mathbf{w} | \theta)\, \mathrm{P}(\theta)$$

- Variational Bayes assumes posterior approximately factorizes

$$\mathrm{P}(\mathbf{w}, \mathbf{t}, \theta) \quad \approx \quad Q(\mathbf{t}) Q(\theta)$$

  EM-like iterations using Inside-Outside (Kurihara and Sato 2006)
- Markov Chain Monte Carlo methods construct a Markov chain whose states are samples from $\mathrm{P}(\mathbf{t}, \theta | \mathbf{w})$

Markov chain Monte Carlo

- MCMC algorithms define a Markov chain where:
  - the states s are the objects we wish to sample; e.g., $s = (\mathbf{t}, \theta)$
    - the state space is astronomically large
  - transition probabilities $\mathrm{P}(s'|s)$ are chosen so that chain converges on desired distribution $\pi(s)$
    - many standard recipes for defining $\mathrm{P}(s'|s)$ from $\pi(s)$ (e.g., Gibbs, Metropolis-Hastings)
- "Run" the chain by:
  - pick a start state s_0
  - pick state s_{t+1} by sampling from $\mathrm{P}(s'|s_t)$
- To estimate the *expected value* of any function f of state s (e.g., rule probabilities θ):
  - discard first few "burn-in" samples from chain
  - average $f(s)$ over the remaining samples from chain

Outline

A Gibbs sampler for $\mathbf{t}$ and θ

- Gibbs samplers require states factor into components $s = (\mathbf{t}, \theta)$
- Update each component in turn by resampling, conditioned on values for other components
  - Resample trees $\mathbf{t}$ given strings $\mathbf{w}$ and rule probabilities θ
  - Resample rule probabilities θ given trees $\mathbf{t}$ and priors α

$$\begin{tikzcd}
 & \alpha_{A_1} \arrow[d] & \cdots & \alpha_{A_j} \arrow[d] & \cdots & \alpha_{A_{|N|}} \arrow[d] & \\
 & \theta_{A_1} \arrow[dl] \arrow[drr] \arrow[drrrrr] & \cdots & \theta_{A_j} \arrow[dlll] \arrow[d] \arrow[drrr] & \cdots & \theta_{A_{|N|}} \arrow[dlllll] \arrow[dll] \arrow[dr] & \\
t_1 \arrow[d] & \cdots & & t_i \arrow[d] & & \cdots & t_n \arrow[d] \\
w_1 & \cdots & & w_i & & \cdots & w_n
\end{tikzcd}$$

$$\begin{aligned}
\mathrm{P}(\mathbf{t}|\theta, \mathbf{w}, \alpha) &= \prod_{i=1}^{n} \mathrm{P}(t_i|w_i, \theta) \\
\mathrm{P}(\theta|\mathbf{t}, \mathbf{w}, \alpha) &= \mathrm{P}_D(\theta|\mathbf{f}(\mathbf{t}) + \alpha) \\
&= \prod_{A \in N} \mathrm{P}_D(\theta|\mathbf{f}_A(\mathbf{t}) + \alpha_A
\end{aligned}$$

- Standard algorithms for sampling these distributions
- Trees $\mathbf{t}$ are *independent given rule probabilities θ*

  $\Rightarrow$ each t_i can be sampled in parallel

  $\Rightarrow$ t_i only influences t_j via θ ("mixes slowly", "poor mobility")

Outline

Marginalizing out the rule probabilities θ

- Define MCMC sampler whose states are the vectors of trees $\mathbf{t}$
- Integrate out the rule probabilities θ, collapsing dependencies and coupling trees

$$\mathrm{P}(\mathbf{t}|\alpha) = \int_{\Delta} \mathrm{P}(\mathbf{t}|\theta)\,\mathrm{P}(\theta|\alpha)\,d\theta = \prod_{A \in N} \frac{Z(\mathbf{f}_A(\mathbf{t}) + \alpha_A)}{Z(\alpha_A)}$$

- Components of state are the trees t_i for strings w_i
  - resample t_i given trees $\mathbf{t}_{-i}$ for other strings $\mathbf{w}_i$

$$\mathrm{P}(t_i|\mathbf{t}_{-i}, \alpha) = \frac{\mathrm{P}(\mathbf{t}|\alpha)}{\mathrm{P}(\mathbf{t}_{-i}|\alpha)} = \prod_{A \in N} \frac{Z(\mathbf{f}_A(\mathbf{t}) + \alpha_A)}{Z(\mathbf{f}_A(\mathbf{t}_{-i}) + \alpha_A)}$$

- (Sample θ from $\mathrm{P}(\theta|\mathbf{t}, \alpha)$ if required).
- If we could sample from

$$\mathrm{P}(t_i|w_i, \mathbf{t}_{-i}, \alpha) = \frac{\mathrm{P}(w_i|t_i)\mathrm{P}(t_i|\mathbf{t}_{-i}, \alpha)}{\mathrm{P}(w_i|\mathbf{t}_{-i}, \alpha)}$$

  we could build a Gibbs sampler whose states are trees $\mathbf{t}$

Why Metropolis-Hastings?

$$\mathrm{P}(t_i | \mathbf{t}_{-i}, \alpha) \quad = \quad \prod_{A \in N} \frac{Z(\mathbf{f}_A(\mathbf{t}) + \alpha_A)}{Z(\mathbf{f}_A(\mathbf{t}_{-i}) + \alpha_A)}$$

- What makes $\mathrm{P}(t_i | \mathbf{t}_{-i}, \alpha)$ so hard to sample?
  - Probability of choosing rule r used n_r times before $\propto n_r + \alpha_r$
  - Previous occurences of r “prime” the rule r
  - *Rule probabilities can change on the fly* inside a sentence
  - Breaks dynamic programming sampling algorithms, which require “context-freeness”
- Metropolis-Hastings algorithms don’t need samples from $\mathrm{P}(t_i | \mathbf{t}_{-i}, \alpha)$
  - sample from a user-specified *proposal distribution* Q
  - use *acceptance-rejection procedure* to convert stream of samples from Q into stream of samples from $\mathrm{P}(\mathbf{t})$
- Proposal distribution Q can be any strictly positive distribution
  - more efficient (fewer rejections) if Q close to $\mathrm{P}(\mathbf{t})$
  - our proposal distribution $Q_i(t_i)$ is *PCFG approximation* $\mathrm{E}[\theta | \mathbf{t}_{-i}, \alpha]$

Metropolis-Hastings collapsed PCFG sampler

- Sampler state: vector of trees $\mathbf{t}$, t_i is a parse of w_i
- Repeat until convergence:
  - randomly choose index i of tree to resample
  - compute PCFG probabilities to be used as proposal distribution

$$\tilde{\theta}_{A\to\beta} \quad = \quad E[\theta_{A\to\beta}|\mathbf{t}_{-i},\alpha] \;=\; \frac{f_{A\to\beta}(\mathbf{t}_{-i}) + \alpha_{A\to\beta}}{\sum_{A\to\beta'\in R_A} f_{A\to\beta'}(\mathbf{t}_{-i}) + \alpha_{A\to\beta'}}$$

  - sample a proposal tree t'_i from $\mathrm{P}(t_i|w_i,\tilde{\theta})$
  - compute acceptance probability $A(t_i, t'_i)$ for t'_i

$$A(t_i, t'_i) \quad = \quad \min\left\{1, \frac{\mathrm{P}(t'_i|\mathbf{t}_{-i},\alpha)\mathrm{P}(t_i|w_i,\tilde{\theta})}{\mathrm{P}(t_i|\mathbf{t}_{-i},\alpha)\mathrm{P}(t'_i|w_i,\tilde{\theta})}\right\}$$

(easy to compute since t'_i is fixed)

  - choose a random number $x \in U[0,1]$
    - if $x < A(t_i, t'_i)$ then *accept* t'_i, i.e., replace t_i with t'_i
    - if $x > A(t_i, t'_i)$ then *reject* t'_i, i.e., keep t_i unchanged

Outline

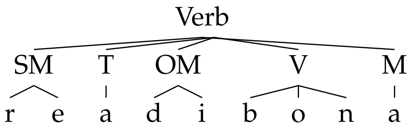
Sesotho verbal morphology

- Sesotho is a Bantu language with complex morphology, not “messed up” much by phonology

re	*a*	*di*	*bon*	*a*
SM	T	OM	V	M

“We see them”

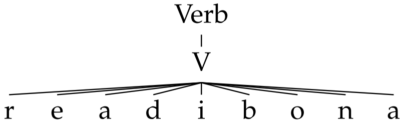
- Demuth’s Sesotho corpus contains morphological parses for 2,283 distinct verb types; can we *learn them automatically*?
- Morphological structure reasonably well described by a CFG



Verb	$\rightarrow$	V
Verb	$\rightarrow$	V M
Verb	$\rightarrow$	SM V M
Verb	$\rightarrow$	SM T V M
Verb	$\rightarrow$	SM T OM V M

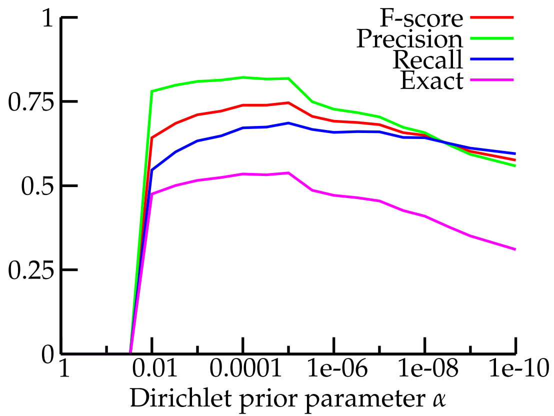
Maximum likelihood finds trivial “saturated” grammar

- Grammar has more productions (81,000) than training strings (2,283)
- Maximum likelihood (e.g., Inside/Outside, EM) tries to make predicted probabilities match empirical probabilities
- *“Saturated” grammar:* every word type has its own production



  - exactly matches empirical probabilities
  - this is what Inside-Outside EM finds
  - none of these analyses are correct

Bayesian estimates with sparse prior find nontrivial structure



- Dirichlet prior for all rules set to same value α
- Dirichlet prior prefers sparse grammars when $\alpha \ll 1$
- Non-trivial structure emerges when $\alpha < 0.01$
- Exact word match accuracy ≈ 0.54 at $\alpha = 10^{-5}$

Outline

Conclusion and future work

- Bayesian estimates incorporate *prior* as well as likelihood
  - product of Dirichlets is conjugate prior for PCFGs
  - can be used to prefer *sparse grammars*
- Even though the full Bayesian posterior is mathematically and computationally intractible, it can be approximated using MCMC
  - Gibbs sampler alternates sampling from $\mathrm{P}(\mathbf{t}|\theta)$ and $\mathrm{P}(\theta|\mathbf{t})$
  - Metropolis-Hastings collapsed sampler integrates out θ and samples $\mathrm{P}(t_i|\mathbf{t}_{-i})$
  - C++ implementations available on Brown web site
- Need to compare these methods with Variational Bayes
- MCMC methods are usually more flexible than other approaches
  - should generalize well to more complex models

Summary

- Context-free grammars are a simple model of hierarchical structure
- The number of trees often grows exponentially with sentence length (depends on grammar)
- Even so, we can find the most likely parse and the probability of a string in cubic time
- Maximum likelihood estimation from trees is easy (relative frequency estimator)
- The Expectation-Maximization algorithm can be used to estimate production probabilities from strings
- The Inside-Outside algorithm calculates the expectations needed for EM in cubic time