Learning rules with Adaptor Grammars
(the Berkeley edition)

Mark Johnson

joint work with Sharon Goldwater and Tom Griffiths

May, 2008
Late one night, a drunk guy is crawling around under a lamppost. A cop comes up and asks him what he’s doing.

“I’m looking for my keys,” the drunk says. “I lost them about three blocks away.”

“So why aren’t you looking for them where you dropped them?” the cop asks.

The drunk looks at the cop, amazed that he’d ask so obvious a question. “Because the light is better here.”
Ideas behind talk

- Most successful statistical learning methods are *parametric*
  - PCFGs have one probability parameter per rule
  - PCFG learning: given rules and data, learn rule probabilities
- Non-parametric learning: learn parameters (rules) as well as values
- *Adaptor grammars:*
  - are a framework for specifying hierarchical nonparametric Bayesian models
  - can express a variety of linguistically-interesting structures
  - are approximated by PCFGs, where number of rules depends on data
  - attempt to put ideas behind Goldwater’s models into a grammatical framework
Language acquisition as Bayesian inference

\[ P(\text{Grammar} \mid \text{Data}) \propto P(\text{Data} \mid \text{Grammar}) \cdot P(\text{Grammar}) \]

- Posterior
- Likelihood
- Prior

- Likelihood measures how well grammar describes data
- Prior expresses knowledge of grammar before data is seen
  - can be very specific (e.g., Universal Grammar)
  - can be very general (e.g., prefer shorter grammars)
- Posterior is *distribution* over grammars
  - expresses uncertainty about which grammar is correct
- But: *infinitely many* grammars may be consistent with Data
Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion
Probabilistic context-free grammars

- Rules in *Context-Free Grammars* (CFGs) expand nonterminals into sequences of terminals and nonterminals
- A *Probabilistic CFG* (PCFG) associates each nonterminal with a multinomial distribution over the rules that expand it
- Probability of a tree is the *product of the probabilities of the rules* used to construct it

<table>
<thead>
<tr>
<th>Rule</th>
<th>$\theta_r$</th>
<th>Rule</th>
<th>$\theta_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP \ VP$</td>
<td>1.0</td>
<td>$NP \rightarrow Sam$</td>
<td>0.75</td>
</tr>
<tr>
<td>$NP \rightarrow Sam$</td>
<td>0.75</td>
<td>$NP \rightarrow Sandy$</td>
<td>0.25</td>
</tr>
<tr>
<td>$VP \rightarrow barks$</td>
<td>0.6</td>
<td>$VP \rightarrow snores$</td>
<td>0.4</td>
</tr>
</tbody>
</table>

\[
P\left(\begin{array}{c}
S \\
\big| \\
NP \\
\big| \\
barks
\end{array}\right) = 0.45 \\
P\left(\begin{array}{c}
S \\
\big| \\
NP \\
\big| \\
Sandy \\
\big| \\
barks
\end{array}\right) = 0.1
\]
Learning syntactic structure is hard

- Bayesian PCFG estimation works well on toy data
- Results are disappointing on “real” data
  - wrong data?
  - wrong rules?
    (rules in PCFG are given a priori; can we learn them too?)
- Strategy: study simpler cases
  - Morphological segmentation (e.g., *walking* = *walk*+*ing*)
  - Word segmentation of unsegmented utterances
A CFG for stem-suffix morphology

- Grammar’s trees can represent any segmentation of words into stems and suffixes
- Can represent true segmentation
- But grammar’s units of generalization (PCFG rules) are “too small” to learn morphemes
A “CFG” with one rule per possible morpheme

Word $\rightarrow$ Stem Suffix
Stem $\rightarrow$ all possible stems
Suffix $\rightarrow$ all possible suffixes

- A rule for each morpheme
  ⇒ “PCFG” can represent probability of each morpheme

- Unbounded number of possible rules, so this is not a PCFG
  - not a practical problem, as only a finite set of rules could possibly be used in any particular data set
Maximum likelihood estimate for $\theta$ is trivial

- Maximum likelihood selects $\theta$ that minimizes KL-divergence between model and training data $\mathcal{W}$ distributions

- *Saturated model* in which each word is generated by its own rule replicates training data distribution $\mathcal{W}$ exactly

$\Rightarrow$ Saturated model is maximum likelihood estimate

- Maximum likelihood estimate does not find any suffixes

```
Word
  Stem   Suffix
    # talking #
```

Forcing generalization via sparse Dirichlet priors

- Idea: use Bayesian prior that prefers fewer rules
- Set of rules is given a priori in Bayesian PCFGs, but can “turn rule off” by setting $\theta_{A \rightarrow \beta} \approx 0$
- Dirichlet prior with $\alpha_{A \rightarrow \beta} \approx 0$ prefers $\theta_{A \rightarrow \beta} \approx 0$

\[ \alpha = (0.1, 1) \]
\[ \alpha = (0.2, 1) \]
\[ \alpha = (0.5, 1) \]
\[ \alpha = (1, 1) \]
Morphological segmentation experiment

- Trained on orthographic verbs from U Penn. Wall Street Journal treebank
- Uniform Dirichlet prior prefers sparse solutions as $\alpha \to 0$
- Metropolis-within-Gibbs sampler used to sample from posterior distribution of parses
  - reanalyses each word based on a grammar estimated from the parses of the other words
### Posterior samples from WSJ verb tokens

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\alpha = 0.1$</th>
<th>$\alpha = 10^{-5}$</th>
<th>$\alpha = 10^{-10}$</th>
<th>$\alpha = 10^{-15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>expect</strong></td>
<td>expect</td>
<td>expect</td>
<td>expect</td>
<td>expect</td>
</tr>
<tr>
<td><strong>expects</strong></td>
<td>expects</td>
<td>expects</td>
<td>expects</td>
<td>expects</td>
</tr>
<tr>
<td><strong>expected</strong></td>
<td>expected</td>
<td>expected</td>
<td>expected</td>
<td>expected</td>
</tr>
<tr>
<td><strong>expecting</strong></td>
<td>expect ing</td>
<td>expect ing</td>
<td>expect ing</td>
<td>expect ing</td>
</tr>
<tr>
<td><strong>include</strong></td>
<td>include</td>
<td>include</td>
<td>include</td>
<td>include</td>
</tr>
<tr>
<td><strong>includes</strong></td>
<td>includes</td>
<td>includes</td>
<td>includes ed</td>
<td>includes ed</td>
</tr>
<tr>
<td><strong>included</strong></td>
<td>included</td>
<td>included</td>
<td>including</td>
<td>including</td>
</tr>
<tr>
<td><strong>including</strong></td>
<td>including</td>
<td>add</td>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>adds</td>
<td>adds</td>
<td>adds</td>
<td>add s</td>
</tr>
<tr>
<td><strong>adds</strong></td>
<td>added</td>
<td>added</td>
<td>add ed</td>
<td>added</td>
</tr>
<tr>
<td><strong>added</strong></td>
<td>adding</td>
<td>adding</td>
<td>add ing</td>
<td>adding</td>
</tr>
<tr>
<td><strong>adding</strong></td>
<td>continue</td>
<td>continue</td>
<td>continue s</td>
<td>continue s</td>
</tr>
<tr>
<td><strong>continue</strong></td>
<td>continues</td>
<td>continues</td>
<td>continue ed</td>
<td>continue ed</td>
</tr>
<tr>
<td><strong>continues</strong></td>
<td>continued</td>
<td>continued</td>
<td>continuing</td>
<td>continuing</td>
</tr>
<tr>
<td><strong>continued</strong></td>
<td>continuing</td>
<td>continu ing</td>
<td>continu ing</td>
<td>continu ing</td>
</tr>
<tr>
<td><strong>continuing</strong></td>
<td>report</td>
<td>report</td>
<td>report</td>
<td>report</td>
</tr>
</tbody>
</table>
Log posterior of models on token data

- Correct solution is nowhere near as likely as posterior
  ⇒ model is wrong!
Independence assumptions in PCFGs

- Context-free grammars are “context-free” because the possible expansions of each node do not depend on expansions of other nodes.
- Probabilistic CFGs extend this by requiring each node expansion to be *statistically independent* (conditioned on the node’s label).
- This is a very strong assumption, which is often false!
- Morphology grammar contains rule:

  \[
  \text{Word} \rightarrow \text{Stem Suffix}
  \]

- Corresponding independence assumption:

  \[
  \Pr(\text{Word}) = \Pr(\text{Stem}) \Pr(\text{Suffix})
  \]

causes PCFG model of morphology to fail.
Relative frequencies of inflected verb forms

![Graph showing relative frequencies of inflected verb forms for different suffixes.](image)
Types and tokens

- A word *type* is a distinct word shape
- A word *token* is an occurrence of a word

Data = “the cat chased the other cat”

Tokens = “the”, “cat”, “chased”, “the”, “other”, “cat”

Types = “the”, “cat”, “chased”, “other”

- Estimating $\theta$ from *word types* rather than word tokens eliminates (most) frequency variation
  - 4 common verb suffixes, so when estimating from verb types $\theta_{\text{Suffix} \rightarrow \text{ing} #} \approx 0.25$

- Several psycholinguists believe that humans learn morphology from word types

- Goldwater et al investigated a morphology-learning model that learnt from an interpolation of types and tokens
### Posterior samples from WSJ verb *types*

<table>
<thead>
<tr>
<th></th>
<th>(\alpha = 0.1)</th>
<th>(\alpha = 10^{-5})</th>
<th>(\alpha = 10^{-10})</th>
<th>(\alpha = 10^{-15})</th>
</tr>
</thead>
<tbody>
<tr>
<td>expect</td>
<td>expect</td>
<td>expect</td>
<td>expect</td>
<td>expect</td>
</tr>
<tr>
<td>expects</td>
<td>expects</td>
<td></td>
<td>expect</td>
<td>expects</td>
</tr>
<tr>
<td>expected</td>
<td>expected</td>
<td>expected</td>
<td></td>
<td>expected</td>
</tr>
<tr>
<td>expect  ing</td>
<td>expect  ing</td>
<td>expect  ing</td>
<td></td>
<td>expect  ing</td>
</tr>
<tr>
<td>include</td>
<td>include</td>
<td>includ e</td>
<td>includ e</td>
<td>includ e</td>
</tr>
<tr>
<td>include s</td>
<td>include es</td>
<td>includ es</td>
<td></td>
<td>includ es</td>
</tr>
<tr>
<td>included</td>
<td>included</td>
<td>includ ed</td>
<td>includ ed</td>
<td>includ ed</td>
</tr>
<tr>
<td>including</td>
<td>including</td>
<td>includ ing</td>
<td>includ ing</td>
<td>includ ing</td>
</tr>
<tr>
<td>add</td>
<td>add</td>
<td>add</td>
<td>add</td>
<td>add</td>
</tr>
<tr>
<td>adds</td>
<td>add s</td>
<td>add s</td>
<td>add s</td>
<td>add s</td>
</tr>
<tr>
<td>add ed</td>
<td>add ed</td>
<td>add ed</td>
<td>add ed</td>
<td>add ed</td>
</tr>
<tr>
<td>adding</td>
<td>add ing</td>
<td>add ing</td>
<td>add ing</td>
<td>add ing</td>
</tr>
<tr>
<td>continue</td>
<td>continue</td>
<td>continu e</td>
<td>continu e</td>
<td>continu e</td>
</tr>
<tr>
<td>continue s</td>
<td>continu es</td>
<td>continu es</td>
<td></td>
<td>continu es</td>
</tr>
<tr>
<td>continu ed</td>
<td>continu ed</td>
<td>continu ed</td>
<td></td>
<td>continu ed</td>
</tr>
<tr>
<td>continuing</td>
<td>continuing</td>
<td>continu ing</td>
<td>continu ing</td>
<td>continu ing</td>
</tr>
<tr>
<td>report</td>
<td>report</td>
<td>repo rt</td>
<td></td>
<td>repo rt</td>
</tr>
</tbody>
</table>
Log posterior of models on type data

- Correct solution is close to optimal at $\alpha = 10^{-3}$
Desiderata for an extension of PCFGs

- PCFG rules are “too small” to be effective units of generalization
  ⇒ generalize over groups of rules
  ⇒ units of generalization should be chosen based on data
- Type-based inference mitigates non-context-free dependencies
  ⇒ Hierarchical Bayesian model where:
    - context-free rules generate types
    - another process replicates types to produce tokens
- *Adaptor grammars:*
  - learn probability of entire subtrees (how a nonterminal expands to terminals)
  - use grammatical hierarchy to define a Bayesian hierarchy, from which type-based inference emerges
  - inspired by Goldwater’s work
Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion
Dirichlet-Multinomials with many outcomes

- Dirichlet prior $\alpha$, observed data $X = (X_1, \ldots, X_n)$
  $$P(X_{n+1} = k \mid X, \alpha) \propto \alpha_k + n_k(X)$$

- Consider a sequence of Dirichlet-multinomials where:
  - total Dirichlet pseudocount is fixed $\alpha_* = \sum_{k=1}^{m} \alpha_k$, and
  - prior uniform over outcomes 1, $\ldots$, $m$, so $\alpha_k = \alpha_*/m$
  - number of outcomes $m \to \infty$

  $$P(X_{n+1} = k \mid X, \alpha_*) \propto \begin{cases} n_k(X) & \text{if } n_k(X) > 0 \\ \alpha_*/m & \text{if } n_k(X) = 0 \end{cases}$$

  But when $m \gg n$, most $k$ are unoccupied (i.e., $n_k(X) = 0$)

  $\Rightarrow$ Probability of a previously seen outcome $k \propto n_k(X)$

  Probability of an outcome never seen before $\propto \alpha_*$
Labeled Chinese restaurant processes (1a)

Generated sequence:

- Each occupied “table” has a label (a “dish”), sampled from $P_B$
- Customer $n + 1$ enters with tables 1, ..., $m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto n_k$
  - sits at new table $k = m + 1$ with probability $\propto \alpha$
- Emit label (dish) on table
  - if table doesn’t have a label, generate one from $P_B$
  $\Rightarrow$ only pay probability cost for label once per table
Labeled Chinese restaurant processes (1b)

Generated sequence: dog

- Each occupied “table” has a label (a “dish”), sampled from $P_B$
- Customer $n + 1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto n_k$
  - sits at new table $k = m + 1$ with probability $\propto \alpha$
- Emit label (dish) on table
  - if table doesn’t have a label, generate one from $P_B$
  ⇒ only pay probability cost for label once per table
Labeled Chinese restaurant processes (2a)

Generated sequence: dog

- Each occupied “table” has a label (a “dish”), sampled from $P_B$
- Customer $n + 1$ enters with tables 1, . . . , $m$ occupied:
  - sits at old table $k \leq m$ with probability $\alpha \cdot n_k$
  - sits at new table $k = m + 1$ with probability $\alpha$
- Emit label (dish) on table
  - if table doesn’t have a label, generate one from $P_B$
  $\Rightarrow$ only pay probability cost for label once per table
**Labeled Chinese restaurant processes (2b)**

Generated sequence: dog, cat

- Each occupied “table” has a label (a “dish”), sampled from $P_B$
- Customer $n+1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto n_k$
  - sits at new table $k = m + 1$ with probability $\propto \alpha$
- Emit label (dish) on table
  - if table doesn’t have a label, generate one from $P_B$
  $\Rightarrow$ only pay probability cost for label *once per table*
Labeled Chinese restaurant processes (3a)

**Generated sequence**: dog, cat

- Each occupied “table” has a label (a “dish”), sampled from $P_B$
- Customer $n + 1$ enters with tables 1, ..., $m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto n_k$
  - sits at new table $k = m + 1$ with probability $\propto \alpha$
- Emit label (dish) on table
  - if table doesn’t have a label, generate one from $P_B$
  \[\Rightarrow\] only pay probability cost for label *once per table*
Labeled Chinese restaurant processes (3b)

Generated sequence: dog, cat, dog

- Each occupied “table” has a label (a “dish”), sampled from $P_B$
- Customer $n + 1$ enters with tables $1, \ldots, m$ occupied:
  - sits at old table $k \leq m$ with probability $\propto n_k$
  - sits at new table $k = m + 1$ with probability $\propto \alpha$
- Emit label (dish) on table
  - if table doesn’t have a label, generate one from $P_B$
    $\Rightarrow$ only pay probability cost for label once per table
From Chinese restaurants to Dirichlet processes

- Chinese restaurant processes map a distribution $P_B$ to a stream of samples from a different distribution with the same support.
- CRPs specify the conditional distribution of the next outcome given the previous ones.
- Each CRP run can produce a different distribution over labels.
- It defines a mapping from $\alpha$ and $P_B$ to a distribution over distributions $DP(\alpha, P_B)$.
- $DP(\alpha, P_B)$ is called a Dirichlet process (DP) with concentration parameter $\alpha$ and base distribution $P_B$.
- The base distribution $P_B$ can be defined by a DP ⇒ hierarchy of DPs.
Nonparametric extensions of PCFGs

- Chinese restaurant processes are a nonparametric extension of Dirichlet-multinomials because the number of states (occupied tables) depends on the data

- Two obvious nonparametric extensions of PCFGs:
  - let the number of nonterminals grow unboundedly
    - refine the nonterminals of an original grammar
e.g., $S_{35} \rightarrow NP_{27} VP_{17}$
    $\Rightarrow$ infinite PCFG
  - let the number of rules grow unboundedly
    - “new” rules are compositions of several rules from original grammar
    - equivalent to caching tree fragments
    $\Rightarrow$ adaptor grammars

- No reason both can’t be done together . . .
Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion
Adaptor grammars: informal description

- An adaptor grammar has a set of CFG rules
- These determine the possible structures as in a CFG
- A subset of the nonterminals are adapted
- Unadapted nonterminals expand by picking a rule and recursively expanding its children, as in a PCFG
- Adapted nonterminals can expand in two ways:
  - by picking a rule and recursively expanding its children, or
  - by generating a previously generated tree (with probability proportional to the number of times previously generated)
- Each adapted subtree behaves like a new rule added to the grammar
- The CFG rules of the adapted nonterminals determine the base distribution over these trees
Adaptor grammars as generative processes

- The sequence of trees generated by an adaptor grammar are not independent
  - it learns from the trees it generates
  - if an adapted subtree has been used frequently in the past, it’s more likely to be used again
- (but the sequence of trees is exchangable)
- An unadapted nonterminal $A$ expands using $A \rightarrow \beta$ with probability $\theta_{A \rightarrow \beta}$
- An adapted nonterminal $A$ expands:
  - to a subtree $\tau$ rooted in $A$ with probability proportional to the number of times $\tau$ was previously generated
  - using $A \rightarrow \beta$ with probability proportional to $\alpha_A \theta_{A \rightarrow \beta}$
Adaptor grammar morphology example

- **Stem** and **Suffix** rules generate all possible stems and suffixes.
- Adapt **Word**, **Stem** and **Suffix** nonterminals.
- One Chinese Restaurant process per adapted nonterminal.

```
Word → Stem Suffix
Stem → # Chars
Suffix → #
Suffix → Chars #
Chars → Char
Chars → Char Chars
Char → a | . . . | z
```
Morphology adaptor grammar (0)

**Word**
- **restaurant**
  - Word $\rightarrow$ Stem \texttt{Suffix}$^*$

**Stem**
- **restaurant**
  - Stem $\rightarrow $ \\
  - Stem $\rightarrow $ \texttt{# \texttt{Chars}}

**Suffix**
- **restaurant**
  - Suffix $\rightarrow $ \\
  - Suffix $\rightarrow $ \texttt{Chars #}

**Chars factory**
- Chars $\rightarrow $ Char
- Chars $\rightarrow $ Char \texttt{Chars}
- Char $\rightarrow $ \texttt{a...z}
Morphology adaptor grammar (1a)

**Word** restaurant
Word → Stem Suffix

**Stem** restaurant
Stem → #
Stem → # Chars

**Suffix** restaurant
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (1b)

**Word** restaurant
- Word $\rightarrow$ Stem Suffix

**Stem** restaurant
- Stem $\rightarrow$ #
- Stem $\rightarrow$ # Chars

**Suffix** restaurant
- Suffix $\rightarrow$ #
- Suffix $\rightarrow$ Char #

**Chars factory**
- Char $\rightarrow$ a...z
Morphology adaptor grammar (1c)

**Word** `restaurant`
- `Word → Stem Suffix`

**Stem** `restaurant`
- `Stem → # Chars` (bu y)
- `Stem → #`

**Suffix** `restaurant`
- `Suffix → # Char #` (s)
- `Suffix → Chars #`

**Chars factory**
- `Chars → Char`
- `Chars → Char Chars`
- `Char → a ... z`
Morphology adaptor grammar (1d)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Chars

**Suffix restaurant**
Suffix → #
Suffix → Chars #

**Chars factory**
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (2a)

**Word**  
restaurant  
Word $\rightarrow$ Stem Suffix

**Stem**  
restaurant  
Stem $\rightarrow$ #  
Stem $\rightarrow$ # Chars

**Suffix**  
restaurant  
Suffix $\rightarrow$ #  
Suffix $\rightarrow$ Char #

**Chars factory**  
Chars $\rightarrow$ Char  
Chars $\rightarrow$ Char Chars  
Char $\rightarrow$ a...z
Morphology adaptor grammar (2b)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Char

**Suffix restaurant**
Suffix → #
Suffix → Char #

**Chars factory**
Chars → Char
Chars → Char Char
Char → a...z
Morphology adaptor grammar (2c)

Word restaurant
Word → Stem Suffix
Word → Stem
Stem
Stem → # Chars
Stem → #
Suffix
Suffix → Char #
Suffix → Chars #
Chars factory
Chars → Char
Chars → Char
Chars → Char Char
Char → a...z
Morphology adaptor grammar (2d)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Char

**Suffix restaurant**
Suffix → #
Suffix → Char #

**Chars factory**
Chars → Char
Chars → Char Char
Char → a...z
Morphology adaptor grammar (3)

**Word restaurant**
Word \(\rightarrow\) Stem Suffix

**Stem restaurant**
Stem \(\rightarrow\) 
Stem \(\rightarrow\) # Chars

**Suffix restaurant**
Suffix \(\rightarrow\) #
Suffix \(\rightarrow\) Chars #

**Chars factory**
Chars \(\rightarrow\) Char
Chars \(\rightarrow\) Char Chars
Char \(\rightarrow\) a...z
Morphology adaptor grammar (4a)

Word restaurant
Word → Stem Suffix

Stem restaurant
Stem → #
Stem → # Chars

Suffix restaurant
Suffix → #
Suffix → Chars #

Chars factory
Chars → Char
Chars → Char Chars
Char → a...z
Morphology adaptor grammar (4b)

**Word restaurant**
- \( \text{Word} \rightarrow \text{Stem} \text{Suffix} \)
- \( \text{Stem} \rightarrow \# \text{Chars} \)
- \( \text{Stem} \rightarrow \# \text{Chars} \)
- \( \text{Suffix} \rightarrow \# \text{Chars} \)
- \( \text{Chars} \rightarrow \text{Char} \text{Chars} \)
- \( \text{Char} \rightarrow \text{a...z} \)

**Stem restaurant**
- \( \text{Stem} \rightarrow \# \text{Chars} \)
- \( \text{Stem} \rightarrow \# \text{Chars} \)
- \( \text{Chars} \rightarrow \text{Char} \text{Chars} \)
- \( \text{Char} \rightarrow \text{a...z} \)

**Suffix restaurant**
- \( \text{Suffix} \rightarrow \# \text{Chars} \)
- \( \text{Chars} \rightarrow \text{Char} \text{Chars} \)
- \( \text{Char} \rightarrow \text{a...z} \)
Morphology adaptor grammar (4c)

Word restaurant
Word → Stem Suffix
Word
Stem
Suffix
# Char
Chars
b u y
s
Suffix
# Char
Chars
r u n
s

Stem restaurant
Stem → #
Stem
# Char
Chars
b u y
Stem
# Char
Chars
r u n

Suffix restaurant
Suffix → #
Suffix
Char
#
Chars
s
Suffix
#

Chars factory
Chars → Char
Chars
Char
a...z
Morphology adaptor grammar (4d)

**Word restaurant**
Word → Stem Suffix

**Stem restaurant**
Stem → #
Stem → # Char

**Suffix restaurant**
Suffix → #
Suffix → Char #

**Chars factory**
Chars → Char
Chars → Char Char
Char → a...z
Properties of adaptor grammars

- Possible trees generated by CFG rules but the probability of each adapted tree is estimated separately.
- Probability of a subtree $\tau$ is proportional to:
  - the number of times $\tau$ was seen before $\Rightarrow$ “rich get richer” dynamics (Zipf distributions)
  - plus $\alpha_A$ times prob. of generating it via PCFG expansion

$\Rightarrow$ Useful compound structures can be more probable than their parts.

- PCFG rule probabilities estimated from table labels $\Rightarrow$ learns from types, not tokens $\Rightarrow$ dampens frequency variation
Bayesian hierarchy inverts grammatical hierarchy

- Grammatically, a Word is composed of a Stem and a Suffix, which are composed of Chars.
- To generate a new Word from an adaptor grammar:
  - reuse an old Word, or
  - generate a fresh one from the base distribution, i.e., generate a Stem and a Suffix.
- Lower in the tree ⇒ higher in Bayesian hierarchy.
Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion
Unigram model of word segmentation

- Unigram model: each word is generated independently
- Input is *unsegmented broad phonemic transcription* (Brent)
  Example: y u w a n t t u s i D 6 b u k
- Adaptor for *Word* non-terminal caches previously seen words

\[
\text{Words} \rightarrow \text{Word}^+ \\
\text{Word} \rightarrow \text{Phoneme}^+
\]

- Unigram word segmentation on Brent corpus: 55% token f-score
Unigram model often finds collocations

- Unigram word segmentation model assumes each word is generated independently
- But there are strong inter-word dependencies (collocations)
- Unigram model can only capture such dependencies by analyzing collocations as words (Goldwater 2006)
Unigram word segmentation grammar learnt

• Based on the base grammar rules

\[
\text{Words} \rightarrow \text{Word}\n\]
\[
\text{Word} \rightarrow \text{Phoneme}\n\]

the adapted grammar contains 1,712 rules such as:

15758 \hspace{0.5cm} \text{Words} \rightarrow \text{Word Words}
9791 \hspace{0.5cm} \text{Words} \rightarrow \text{Word}
1660 \hspace{0.5cm} \text{Word} \rightarrow \text{Phoneme}\n402 \hspace{0.5cm} \text{Word} \rightarrow y\ u
137 \hspace{0.5cm} \text{Word} \rightarrow \ i\ n
111 \hspace{0.5cm} \text{Word} \rightarrow w\ l\ T
100 \hspace{0.5cm} \text{Word} \rightarrow D\ 6\ d\ O\ g\ i
45 \hspace{0.5cm} \text{Word} \rightarrow \ l\ n\ D\ 6
20 \hspace{0.5cm} \text{Word} \rightarrow \ l\ n\ D\ 6\ h\ Q\ s
Modeling collocations improves segmentation

- A Colloc(ation) consists of one or more words
- Both Words and Colloc(s) are adapted (learnt)
- Significantly improves word segmentation accuracy over unigram model (75% f-score; ≈ Goldwater’s bigram model)
- Two levels of Collocations improves slightly (76%)
Syllables + Collocations + Word segmentation

Sentence → Colloc⁺
Word → SyllableIF
Word → SyllableI Syllable SyllableF
Onset → Consonant⁺
Nucleus → Vowel⁺

Colloc → Word⁺
Word → SyllableI SyllableF
Syllable → (Onset) Rhyme
Rhyme → Nucleus (Coda)
Coda → Consonant⁺

• With no supra-word generalizations, f-score = 68%
• With 2 Collocation levels, f-score = 84%
• Without distinguishing initial/final clusters, f-score = 82%
Syllables + 2-level Collocations + Word segmentation
Word segmentation results summary

<table>
<thead>
<tr>
<th>Collocation levels above the word</th>
<th>none</th>
<th>1 level</th>
<th>2 levels</th>
<th>3 levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>0.55</td>
<td>0.73</td>
<td>0.75</td>
<td>0.74</td>
</tr>
<tr>
<td>morphemes</td>
<td>0.35</td>
<td>0.55</td>
<td>0.79</td>
<td>0.78</td>
</tr>
<tr>
<td>syllables</td>
<td>0.32</td>
<td>0.69</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td>syllables IF</td>
<td>0.46</td>
<td>0.68</td>
<td>0.84</td>
<td>0.84</td>
</tr>
</tbody>
</table>

- We can learn collocations and syllable structure together with word segmentation, even though we don’t know where the word boundaries are.
- Learning these together improves word segmentation accuracy.
  - Are there other examples of *synergistic interaction* in language learning?
Estimating adaptor grammars

- Need to estimate:
  - cached subtrees $\tau$ for adapted nonterminals
  - (optional) DP parameters $\alpha$ for adapted nonterminals
  - (optional) probabilities $\theta$ of base grammar rules

- Component-wise Metropolis-within-Gibbs sampler
  - components are parse tree $T_i$ for each string $W_i$
  - sample $T_i$ from $P(T|W_i, T_{-i}, \alpha, \theta)$ for each sentence $W_i$ in turn

- Sampling directly from conditional distribution of parses seems intractable
  - construct PCFG proposal grammar $G'(T_{-i})$ on the fly
  - each table label $\tau$ corresponds to a production in PCFG approximation
  - Use accept/reject to convert samples from PCFG approx to samples from adaptor grammar
Metropolis-with-Gibbs sampler

- Collapsed Gibbs sampler: resample parse $T_i$ given $W_i$ and $T_{-i}$
- Table counts change within a parse tree
  - grammar is not context-free
  - breaks standard dynamic programming
  - Metropolis accept/reject for each Gibbs sample
- PCFG can express probability of selecting a table given $T_{-i}$
  - ignores changing table counts within single parse
- Rules of PCFG proposal grammar $G'$ consist of:
  - rules $A \rightarrow \beta$ from base PCFG: $\theta'_{A\rightarrow\beta} \propto \alpha_A \theta_{A\rightarrow\beta}$
  - A rule $A \rightarrow \text{YIELD}(\tau)$ for each table $\tau$ in $A$’s restaurant: $\theta'_{A\rightarrow\text{YIELD}(\tau)} \propto n_\tau$, the number of customers at table $\tau$
- Parses of $G'$ can be mapped back to adaptor grammar parses
Bayesian priors on adaptor grammar parameters

- Parameters of adaptor grammars:
  - probabilities $\theta_{A \rightarrow \beta}$ of base grammar rules $A \rightarrow \beta$
  - concentration parameters $\alpha_A$ of adapted nonterminals $A$

- Put Bayesian priors on these parameters
  - (Uniform) Dirichlet prior on base grammar rule probabilities $\theta$
  - Vague Gamma prior on concentration parameter on $\alpha_A$

- We also use a generalization of CRPs called “Pitman-Yor processes”, and put a uniform Dirichlet prior on its $a$ parameter

- We use a Metropolis-Hastings sampler for $a$ and $b$ parameters
  - $a$ is sampled from sequence of increasingly narrow Dirichlets
  - $b$ is sampled from sequence of increasingly narrow Gammas

- Seems to improve performance with complicated grammars
Random initialization is better than incremental initialization

- Incremental initialization: assign parse for $W_i$ based on $T_{1,i-1}$
- Random initialization: initially assign parses $T_i$ randomly
- Incremental initialization seems to get stuck in local optima
Table label resampling improves mobility

- Gibbs algorithm: resample $T_i$ given $W_i$ and $T_{-i}$
- Table label resampling resamples the labels on each table
  - can change parses for many sentences at once

<table>
<thead>
<tr>
<th>Word</th>
<th>restaurant</th>
<th>Word → Stem Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stem</td>
<td>restaurant</td>
<td>Stem → #</td>
</tr>
<tr>
<td>Suffix</td>
<td>-&gt; #</td>
<td>Suffix → Chars #</td>
</tr>
<tr>
<td>Chars</td>
<td>factory</td>
<td>Chars → Char</td>
</tr>
</tbody>
</table>
Table label resampling with Colloc grammar

- log posterior probability

Iteration

0  200  400  600  800  1000

no resampling
resampling
resampling to iteration 100
Segmentation accuracy with Colloc grammar

- random init, resampling
- random init, no resampling
- random init, resampling to iteration 100
- sequential init, resampling
- sequential init, no resampling
- sequential init, resampling to iteration 100

Word token f-score

<table>
<thead>
<tr>
<th>Iteration</th>
<th>0</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Iteration</th>
<th>0</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>0.8</td>
<td>1.2</td>
</tr>
</tbody>
</table>

Graph showing the performance of different initialization and resampling strategies over iterations.
Outline

Probabilistic Context-Free Grammars

Chinese Restaurant Processes

Adaptor Grammars

Word segmentation with Adaptor Grammars

Bayesian inference for Adaptor Grammars

Extending Adaptor Grammars

Conclusion
Issues with adaptor grammars

- Recursion *through adapted nonterminals* seems problematic
  - New tables are created as each node is encountered top-down
  - But the tree labeling the table is only known after the whole subtree has been completely generated
  - If adapted nonterminals are recursive, might pick a table whose label we are currently constructing. What then?
- Extend adaptor grammars so adapted fragments can end at nonterminals a la DOP (currently always go to terminals)
  - Adding “exit probabilities” to each adapted nonterminal
  - In some approaches, fragments can grow “above” existing fragments, but can’t grow “below” (O’Donnell)
- Adaptor grammars *conflate grammatical and Bayesian hierarchies*
  - Might be useful to disentangle them with *meta-grammars*
Context-free grammars

A context-free grammar (CFG) consists of:
- a finite set $N$ of nonterminals,
- a finite set $W$ of terminals disjoint from $N$,
- a finite set $R$ of rules $A \rightarrow \beta$, where $A \in N$ and $\beta \in (N \cup W)^*$
- a start symbol $S \in N$.

Each $A \in N \cup W$ generates a set $T_A$ of trees. These are the smallest sets satisfying:
- If $A \in W$ then $T_A = \{A\}$.
- If $A \in N$ then:

$$T_A = \bigcup_{A \rightarrow B_1 \ldots B_n \in R_A} \text{TREE}_A(T_{B_1}, \ldots, T_{B_n})$$

where $R_A = \{A \rightarrow \beta : A \rightarrow \beta \in R\}$, and

$$\text{TREE}_A(T_{B_1}, \ldots, T_{B_n}) = \left\{ \frac{A}{t_1 \ldots t_n} : t_i \in T_{B_i}, \quad i = 1, \ldots, n \right\}$$

The set of trees generated by a CFG is $T_S$. 
Probabilistic context-free grammars

A probabilistic context-free grammar (PCFG) is a CFG and a vector $\theta$, where:

- $\theta_{A \rightarrow \beta}$ is the probability of expanding the nonterminal $A$ using the production $A \rightarrow \beta$.

It defines distributions $G_A$ over trees $T_A$ for $A \in N \cup W$:

$$G_A = \begin{cases} 
\delta_A & \text{if } A \in W \\
\sum \limits_{A \rightarrow B_1 \ldots B_n \in R_A} \theta_{A \rightarrow B_1 \ldots B_n} \text{TD}_A(G_{B_1}, \ldots, G_{B_n}) & \text{if } A \in N
\end{cases}$$

where $\delta_A$ puts all its mass onto the singleton tree $A$, and:

$$\text{TD}_A(G_1, \ldots, G_n) \left( \begin{array}{c}
A \\
\overbrace{t_1 \ldots t_n}
\end{array} \right) = \prod_{i=1}^{n} G_i(t_i).$$

$\text{TD}_A(G_1, \ldots, G_n)$ is a distribution over $T_A$ where each subtree $t_i$ is generated independently from $G_i$. 
DP adaptor grammars

An adaptor grammar \( (G, \theta, \alpha) \) is a PCFG \( (G, \theta) \) together with a parameter vector \( \alpha \) where for each \( A \in N \), \( \alpha_A \) is the parameter of the Dirichlet process associated with \( A \).

\[
G_A \sim \text{DP}(\alpha_A, H_A) \text{ if } \alpha_A > 0 \\
= H_A \quad \text{ if } \alpha_A = 0
\]

\[
H_A = \sum_{A \rightarrow B_1 \ldots B_n \in R_A} \theta_{A \rightarrow B_1 \ldots B_n} \text{TD}_A(G_{B_1}, \ldots, G_{B_n})
\]

The grammar generates the distribution \( G_S \).
One Dirichlet Process for each adapted non-terminal \( A \) (i.e., \( \alpha_A > 0 \)).
Recursion in adaptor grammars

- The probability of joint distributions \((G, H)\) is defined by:

\[
G_A \sim \text{DP}(\alpha_A, H_A) \quad \text{if} \quad \alpha_A > 0
\]
\[
= H_A \quad \text{if} \quad \alpha_A = 0
\]

\[
H_A = \sum_{A \rightarrow B_1 \ldots B_n \in R_A} \theta_{A \rightarrow B_1 \ldots B_n} \text{TD}_A(G_{B_1}, \ldots, G_{B_n})
\]

- This holds even if adaptor grammar is recursive
- Question: when does this define a distribution over \((G, H)\)?
Adaptive fragment grammars

- Disentangle syntactic and Bayesian hierarchy
  - Adaptive metagrammar generates fragment distributions
  - which plug together as in tree substitution grammar

- Tree fragment sets $\mathcal{P}_A, A \in \mathcal{N}$ are smallest sets satisfying:

$$\mathcal{P}_A = \bigcup_{A \rightarrow B_1 \ldots B_n \in \mathcal{R}_A} \text{TREE}_A(\{B_1\} \cup \mathcal{P}_{B_1}, \ldots, \{B_n\} \cup \mathcal{P}_{B_n})$$

- Grammar’s distributions $G_A$ over $\mathcal{T}_A$ defined using fragment distributions $F_A$ over $\mathcal{P}_A$ (generalized PCFG rules)

$$G_A = \sum_{\begin{array}{c} \overbrace{A}^{A} \\
\overbrace{B_1 \ldots B_n}^{B_1 \ldots B_n} \end{array} \in \mathcal{P}_A} F_A(\begin{array}{c} \overbrace{A}^{A} \\
\overbrace{B_1 \ldots B_n}^{B_1 \ldots B_n} \end{array}) \quad \text{TD}_A(G_{B_1}, \ldots, G_{B_n})$$

- A fragment grammar generates the distribution $G_S$
Adaptive fragment distributions

- \( H_A \) is a PCFG distribution over \( \mathcal{P}_A \)

\[
H_A = \sum_{A \rightarrow B_1 \ldots B_n \in R_A} \theta_{A \rightarrow B_1 \ldots B_n} \text{TD}_A(\eta \delta_{B_1} + (1 - \eta)H_{B_1}, \ldots)
\]

where \( \eta \) is the fragment exit probability

- Obtain \( F_A \) by adapting the \( H_A \) distribution

\[
F_A \sim \text{DP}(\alpha_A, H_A)
\]

- This construction can be iterated, i.e., replace \( \theta \) with another fragment distribution

- Question: if we iterate this, when does the fixed point exist, and what is it?
Outline

Probabilistic Context-Free Grammars
Chinese Restaurant Processes
Adaptor Grammars
Word segmentation with Adaptor Grammars
Bayesian inference for Adaptor Grammars
Extending Adaptor Grammars

Conclusion
Summary and future work

- Adaptor grammars “adapt” their distribution to the strings they have generated.
- They learn the probabilities of the subtrees of the adapted nonterminals.
- This makes adaptor grammars non-parametric; the subtrees they cache depends on the data.
- A variety of different linguistic phenomena can be described with adaptor grammars.
- Because they are grammars, they are easy to design and compose.
- The basic approach seems quite flexible.
  - many possible extensions of Adaptor Grammars.
- MCMC sampling algorithm may not scale well to large data or complicated grammars. Are there better estimators?