

Learning rules with Adaptor Grammars

Mark Johnson

Macquarie University

joint work with Sharon Goldwater and Tom Griffiths

February 2010

The drunk under the lamppost

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 so much better here.”*

“There exists today a very elaborate system of formal logic, and specifically, of logic as applied to mathematics. This is a discipline with many good sides, but also with certain serious weaknesses. . . .

Everybody who has worked in formal logic will confirm that it is one of the technically most refractory parts of mathematics. The reason for this is that it deals with rigid, all-or-none concepts, and has very little contact with the continuous concept of the real or of complex number, that is, with mathematical analysis. Yet analysis is the technically most successful and best-elaborated part of mathematics.

Thus formal logic is, by the nature of its approach, cut off from the best cultivated portions of mathematics, and forced onto the most difficult part of mathematical terrain, into combinatorics.”

— John von Neumann

Ideas behind talk

- Statistical methods have revolutionized computational linguistics and cognitive science
- But most successful learning methods are *parametric*
 - ▶ learn values of parameters of a *fixed number of elements*
- *Non-parametric Bayesian methods* can learn the elements as well as their weights
- *Adaptor Grammars* use grammars to specify possible elements
 - ▶ Adaptor Grammar learns probability of each *adapted subtree* it generates
 - ▶ simple “*rich get richer*” learning rule
- Applications of Adaptor Grammars:
 - ▶ acquisition of *concatenative morphology*
 - ▶ *word segmentation* (precursor of lexical acquisition)
 - ▶ learning the structure of *named-entity NPs*
- Sampling (instead of EM) is a natural approach to Adaptor Grammar inference

Outline

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

Bayesian inference for a proposition

- Bayesians interpret Bayes rule as a *prescription of how to update beliefs*

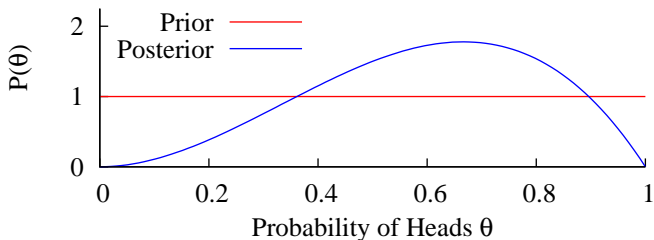
$$\underbrace{P(\text{Hypothesis} \mid \text{Data})}_{\text{Posterior}} \propto \underbrace{P(\text{Data} \mid \text{Hypothesis})}_{\text{Likelihood}} \underbrace{P(\text{Hypothesis})}_{\text{Prior}}$$

- Hypothesis: $Rain$ = “It is raining during this talk”
- Prior: $P(Rain) = 0.5, P(\neg Rain) = 0.5$
- Data: Wet = “Footpath is wet”
- Likelihood: $P(Wet \mid Rain) = 0.8, P(Wet \mid \neg Rain) = 0.4$
- Posterior: $P(Rain \mid Wet) = 2/3, P(\neg Rain \mid Wet) = 1/3$

Bayesian inference for a parameter

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

- Hypothesis: “Probability of coin coming up Heads is θ ”
- Prior: every value for $\theta \in [0, 1]$ is equally likely, i.e., $P(\theta) = 1$
- Data: “Three flips: Heads, Tails, Heads (HTH)”
- Likelihood: $P(HTH \mid \theta) = \theta \cdot (1 - \theta) \cdot \theta$
- Posterior: $P(\theta \mid HTH) \propto \theta^2 \cdot (1 - \theta)$



Language acquisition as Bayesian inference

$$\underbrace{P(\text{Grammar} \mid \text{Data})}_{\text{Posterior}} \propto \underbrace{P(\text{Data} \mid \text{Grammar})}_{\text{Likelihood}} \underbrace{P(\text{Grammar})}_{\text{Prior}}$$

- 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 a *distribution* over grammars
 - ▶ captures *learner's uncertainty* about which grammar is correct
- Grammatical inference is *non-parametric* because we have to learn *how many parameters* there are (e.g., the size of the vocabulary) as well as their values

Outline

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

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

Rule r	θ_r
$S \rightarrow NP VP$	1.0
$NP \rightarrow \text{Sam}$	0.75
$VP \rightarrow \text{barks}$	0.6

Rule r	θ_r
$NP \rightarrow \text{Sandy}$	0.25
$VP \rightarrow \text{snores}$	0.4

$$P \left(\begin{array}{c} S \\ \swarrow \quad \searrow \\ NP \quad VP \\ | \quad | \\ \text{Sam} \quad \text{barks} \end{array} \right) = 0.45$$

$$P \left(\begin{array}{c} S \\ \swarrow \quad \searrow \\ NP \quad VP \\ | \quad | \\ \text{Sandy} \quad \text{snores} \end{array} \right) = 0.1$$

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

Word \rightarrow Stem Suffix

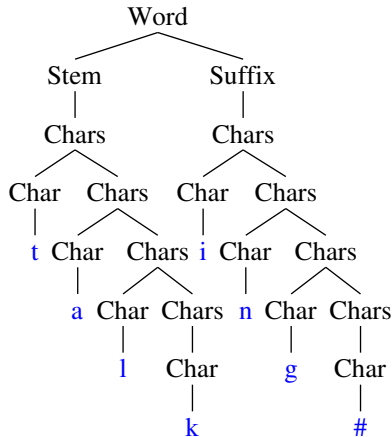
Stem \rightarrow Chars

Suffix \rightarrow Chars

Chars \rightarrow Char

Chars \rightarrow Char Chars

Char \rightarrow a | b | c | ...



- Grammar's trees can represent any segmentation of words into stems and suffixes

\Rightarrow Can *represent* true segmentation

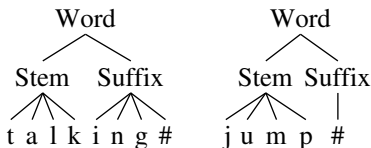
- But grammar's *units of generalization (PCFG rules)* are "too small" to learn morphemes

A “CFG” with one rule per possible morpheme

Word \rightarrow Stem Suffix

Stem \rightarrow *all possible stems*

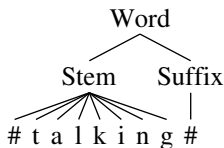
Suffix \rightarrow *all possible suffixes*



- A rule for each morpheme
 \Rightarrow “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

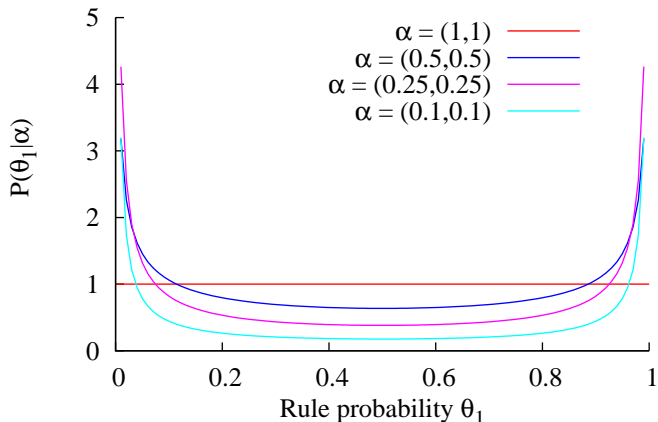
Maximum likelihood estimate for θ is trivial

- Maximum likelihood selects θ that minimizes KL-divergence between model and training data \mathbf{W} distributions
 - *Saturated model* in which each word is generated by its own rule replicates training data distribution \mathbf{W} exactly
- ⇒ Saturated model is maximum likelihood estimate
- Maximum likelihood estimate does not find any suffixes



Forcing generalization via sparse Dirichlet priors

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



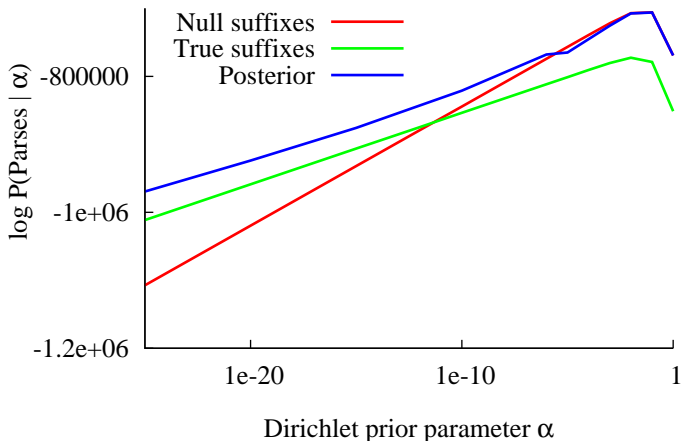
Morphological segmentation experiment

- Trained on orthographic verbs from U Penn. Wall Street Journal treebank
- Uniform Dirichlet prior prefers sparse solutions as $\alpha \rightarrow 0$
- Gibbs sampler samples from posterior distribution of parses
 - ▶ reanalyses each word based on parses of the other words

Posterior samples from WSJ verb tokens

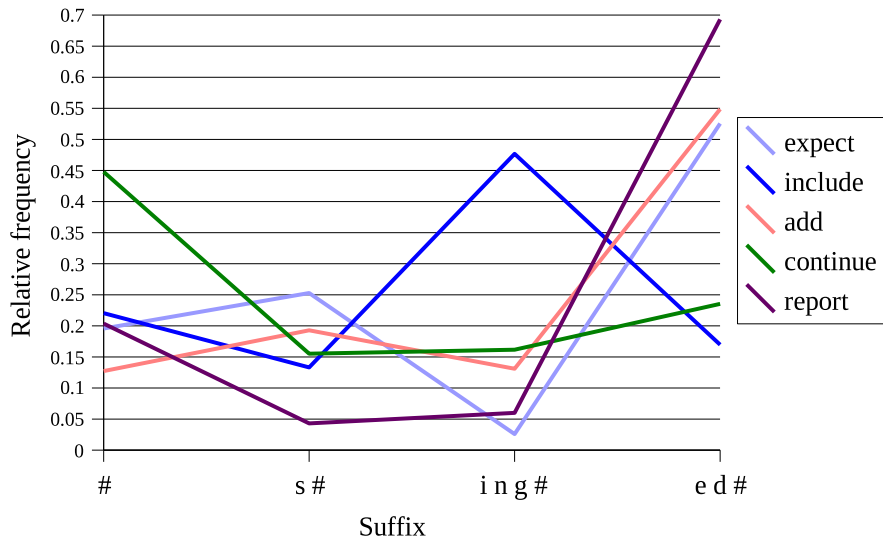
$\alpha = 0.1$	$\alpha = 10^{-5}$	$\alpha = 10^{-10}$	$\alpha = 10^{-15}$
expect	expect	expect	expect
expects	expects	expects	expects
expected	expected	expected	expected
expecting	expect ing	expect ing	expect ing
include	include	include	include
includes	includes	includ es	includ es
included	included	includ ed	includ ed
including	including	including	including
add	add	add	add
adds	adds	adds	add s
added	added	add ed	added
adding	adding	add ing	add ing
continue	continue	continue	continue
continues	continues	continue s	continue s
continued	continued	continu ed	continu ed
continuing	continuing	continu ing	continu ing
report	report	report	report

Log posterior for models on token data



- Correct solution is nowhere near as likely as posterior
 \Rightarrow model is wrong!

Relative frequencies of inflected verb forms



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”

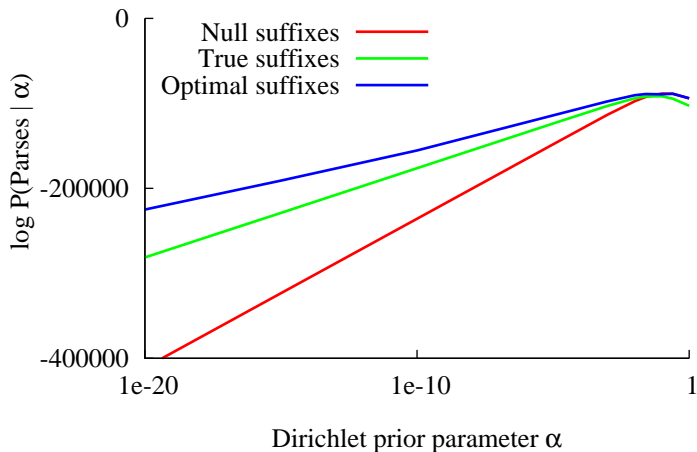
Types = “the”, “cat”, “chased”, “other”

- Estimating θ 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
- Adaptor grammar mimics Goldwater et al “Interpolating between Types and Tokens” morphology-learning model

Posterior samples from WSJ verb *types*

$\alpha = 0.1$	$\alpha = 10^{-5}$	$\alpha = 10^{-10}$	$\alpha = 10^{-15}$
expect	expect	expect	exp ect
expects	expect s	expect s	exp ects
expected	expect ed	expect ed	exp ected
expect ing	expect ing	expect ing	exp ecting
include	includ e	includ e	includ e
include s	includ es	includ es	includ es
included	includ ed	includ ed	includ ed
including	includ ing	includ ing	includ ing
add	add	add	add
adds	add s	add s	add s
add ed	add ed	add ed	add ed
adding	add ing	add ing	add ing
continue	continu e	continu e	continu e
continue s	continu es	continu es	continu es
continu ed	continu ed	continu ed	continu ed
continuing	continu ing	continu ing	continu ing
report	report	repo rt	rep ort

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 over-dispersion
 - ⇒ 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

Outline

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

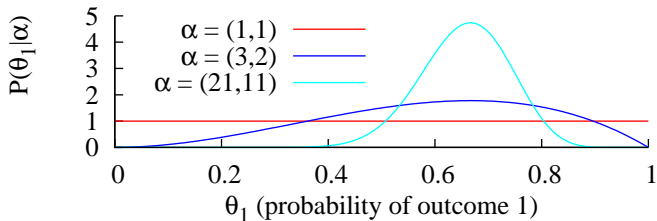
Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

Multinomial and Dirichlet distributions

- A *multinomial* is a distribution over multiple independent trials each with the same finite set of outcomes (e.g., rolls of a die)
 - ▶ specified by vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_m)$,
where outcome $k \in 1, \dots, m$ has probability θ_k
- A *Dirichlet distribution* is a probability distribution over multinomial parameter vectors $\boldsymbol{\theta}$
 - ▶ specified by vector $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_m)$
- If $\boldsymbol{\theta} \sim \text{Dirichlet}(\boldsymbol{\alpha})$ then $P(k \mid \boldsymbol{\alpha}) \propto \alpha_k$
- If *prior* is Dirichlet with parameters $\boldsymbol{\alpha}$, and *data* is $\boldsymbol{n} = (n_1, \dots, n_m)$, where k is seen n_k times then *posterior* is Dirichlet with parameters $\boldsymbol{\alpha} + \boldsymbol{n}$



Dirichlet-Multinomials with many outcomes

- Dirichlet prior $\boldsymbol{\alpha}$, observed data $\mathbf{z} = (z_1, \dots, z_n)$

$$P(Z_{n+1} = k \mid \mathbf{z}, \boldsymbol{\alpha}) \propto \alpha_k + n_k(\mathbf{z})$$

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

$$P(Z_{n+1} = k \mid \mathbf{z}, \alpha) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } n_k(\mathbf{z}) > 0 \\ \alpha/m & \text{if } n_k(\mathbf{z}) = 0 \end{cases}$$

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

- \Rightarrow *Probability of a previously seen outcome $k \propto n_k(\mathbf{z})$*
Probability of an outcome never seen before $\propto \alpha$

From Dirichlet-multinomials to Chinese Restaurant Processes

- Observations $\mathbf{z} = (z_1, \dots, z_n)$ ranging over outcomes $1, \dots, m$
- Outcome k observed $n_k(\mathbf{z})$ times in data \mathbf{z}
- *Predictive distribution* with uniform Dirichlet prior:

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto n_k(\mathbf{z}) + \alpha/m$$

- Let $m \rightarrow \infty$

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto n_k(\mathbf{z}) \text{ if } k \text{ appears in } \mathbf{z}$$

$$P(Z_{n+1} \notin \mathbf{z} \mid \mathbf{z}) \propto \alpha$$

- If outcomes are exchangeable \Rightarrow number in order of occurrence
 \Rightarrow *Chinese Restaurant Process*

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } k \leq m = \max(\mathbf{z}) \\ \alpha & \text{if } k = m + 1 \end{cases}$$

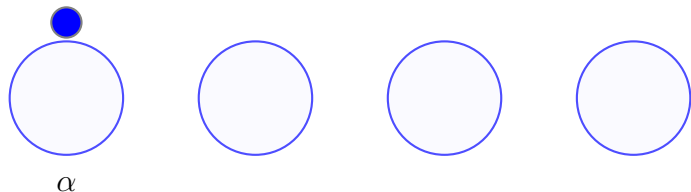
Chinese Restaurant Process (0)



- Customer \rightarrow table mapping $\mathbf{z} =$
- $P(\mathbf{z}) = 1$
- Next customer chooses a table according to:

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } k \leq m = \max(\mathbf{z}) \\ \alpha & \text{if } k = m + 1 \end{cases}$$

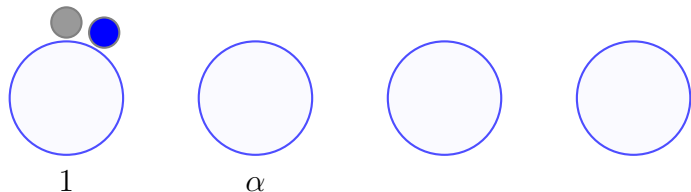
Chinese Restaurant Process (1)



- Customer \rightarrow table mapping $\mathbf{z} = 1$
- $P(\mathbf{z}) = \alpha/\alpha$
- Next customer chooses a table according to:

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } k \leq m = \max(\mathbf{z}) \\ \alpha & \text{if } k = m + 1 \end{cases}$$

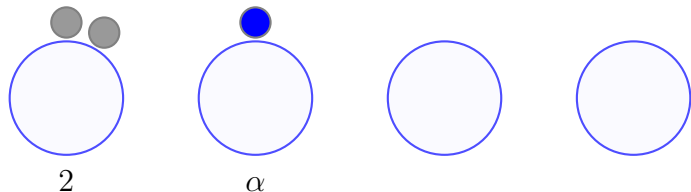
Chinese Restaurant Process (2)



- Customer \rightarrow table mapping $\mathbf{z} = 1, 1$
- $P(\mathbf{z}) = \alpha/\alpha \times 1/(1 + \alpha)$
- Next customer chooses a table according to:

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } k \leq m = \max(\mathbf{z}) \\ \alpha & \text{if } k = m + 1 \end{cases}$$

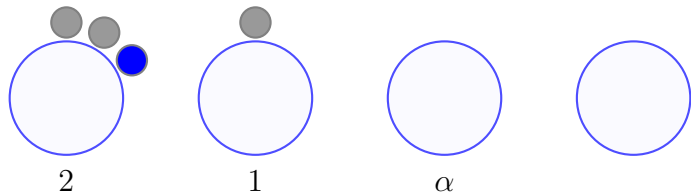
Chinese Restaurant Process (3)



- Customer \rightarrow table mapping $\mathbf{z} = 1, 1, 2$
- $P(\mathbf{z}) = \alpha/\alpha \times 1/(1 + \alpha) \times \alpha/(2 + \alpha)$
- Next customer chooses a table according to:

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } k \leq m = \max(\mathbf{z}) \\ \alpha & \text{if } k = m + 1 \end{cases}$$

Chinese Restaurant Process (4)



- Customer \rightarrow table mapping $\mathbf{z} = 1, 1, 2, 1$
- $P(\mathbf{z}) = \alpha/\alpha \times 1/(1 + \alpha) \times \alpha/(2 + \alpha) \times 2/(3 + \alpha)$
- Next customer chooses a table according to:

$$P(Z_{n+1} = k \mid \mathbf{z}) \propto \begin{cases} n_k(\mathbf{z}) & \text{if } k \leq m = \max(\mathbf{z}) \\ \alpha & \text{if } k = m + 1 \end{cases}$$

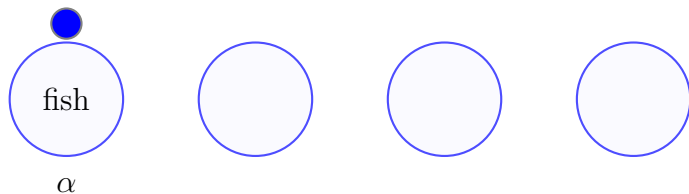
Labeled Chinese Restaurant Process (0)



- Table \rightarrow label mapping $\mathbf{y} =$
- Customer \rightarrow table mapping $\mathbf{z} =$
- Output sequence $\mathbf{x} =$
- $P(\mathbf{x}) = 1$

- *Base distribution* $P_0(Y)$ generates a *label* y_k for each table k
- All customers sitting at table k (i.e., $z_i = k$) share label y_k
- Customer i sitting at table z_i has label $x_i = y_{z_i}$

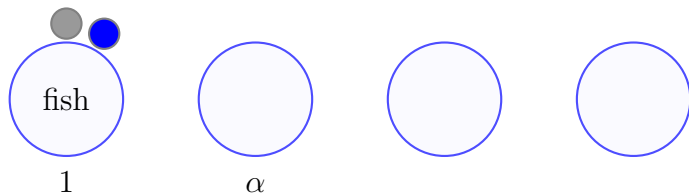
Labeled Chinese Restaurant Process (1)



- Table \rightarrow label mapping $\mathbf{y} = \text{fish}$
- Customer \rightarrow table mapping $\mathbf{z} = 1$
- Output sequence $\mathbf{x} = \text{fish}$
- $P(\mathbf{x}) = \alpha / \alpha \times P_0(\text{fish})$

- *Base distribution* $P_0(Y)$ generates a *label* y_k for each table k
- All customers sitting at table k (i.e., $z_i = k$) share label y_k
- Customer i sitting at table z_i has label $x_i = y_{z_i}$

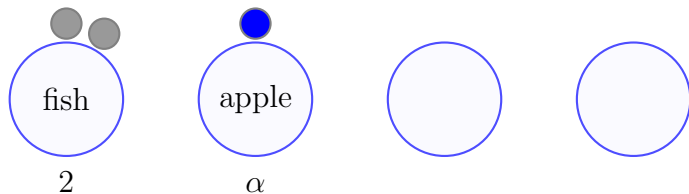
Labeled Chinese Restaurant Process (2)



- Table \rightarrow label mapping $\mathbf{y} = \text{fish}$
- Customer \rightarrow table mapping $\mathbf{z} = 1, 1$
- Output sequence $\mathbf{x} = \text{fish}, \text{fish}$
- $P(\mathbf{x}) = P_0(\text{fish}) \times 1/(1 + \alpha)$

- *Base distribution* $P_0(Y)$ generates a *label* y_k for each table k
- All customers sitting at table k (i.e., $z_i = k$) share label y_k
- Customer i sitting at table z_i has label $x_i = y_{z_i}$

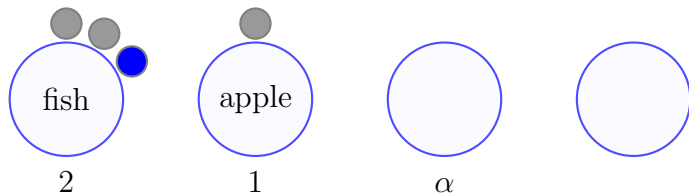
Labeled Chinese Restaurant Process (3)



- Table \rightarrow label mapping $\mathbf{y} = \text{fish, apple}$
- Customer \rightarrow table mapping $\mathbf{z} = 1, 1, 2$
- Output sequence $\mathbf{x} = \text{fish, fish, apple}$
- $P(\mathbf{x}) = P_0(\text{fish}) \times 1/(1 + \alpha) \times \alpha/(2 + \alpha)P_0(\text{apple})$

- *Base distribution* $P_0(Y)$ generates a *label* y_k for each table k
- All customers sitting at table k (i.e., $z_i = k$) share label y_k
- Customer i sitting at table z_i has label $x_i = y_{z_i}$

Labeled Chinese Restaurant Process (4)



- Table \rightarrow label mapping $\mathbf{y} = \text{fish, apple}$
- Customer \rightarrow table mapping $\mathbf{z} = 1, 1, 2$
- Output sequence $\mathbf{x} = \text{fish, fish, apple, fish}$
- $P(\mathbf{x}) = P_0(\text{fish}) \times 1/(1 + \alpha) \times \alpha/(2 + \alpha) P_0(\text{apple}) \times 2/(3 + \alpha)$
- *Base distribution* $P_0(Y)$ generates a *label* y_k for each table k
- All customers sitting at table k (i.e., $z_i = k$) share label y_k
- Customer i sitting at table z_i has label $x_i = y_{z_i}$

Summary: Chinese Restaurant Processes

- *Chinese Restaurant Processes* (CRPs) generalize Dirichlet-Multinomials to an *unbounded number of outcomes*
 - ▶ *concentration parameter* α controls how likely a new outcome is
 - ▶ CRPs exhibit a *rich get richer* power-law behaviour
- *Labeled CRPs* use a *base distribution* to label each table
 - ▶ base distribution can have *infinite support*
 - ▶ concentrates mass on a countable subset
 - ▶ power-law behaviour \Rightarrow Zipfian distributions

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

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

Adaptor grammars: informal description

- The trees generated by an adaptor grammar are defined by CFG rules 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)
- Implemented by having a CRP for each adapted nonterminal
- The CFG rules of the adapted nonterminals determine the *base distributions* of these CRPs

Adaptor grammar for stem-suffix morphology (0)

Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



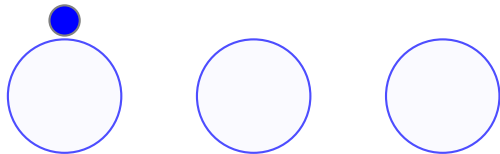
Suffix \rightarrow Phoneme^{*}



Generated words:

Adaptor grammar for stem-suffix morphology (1a)

Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



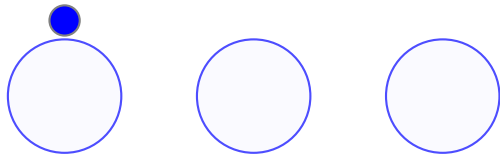
Suffix \rightarrow Phoneme^{*}



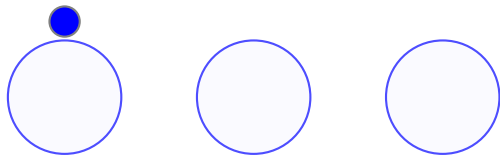
Generated words:

Adaptor grammar for stem-suffix morphology (1b)

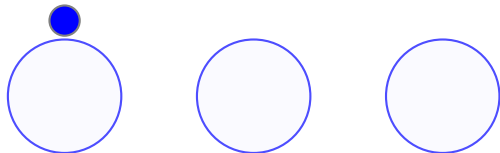
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



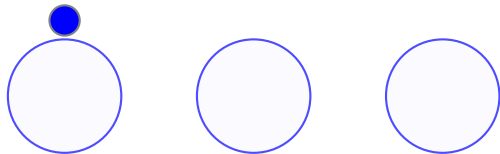
Suffix \rightarrow Phoneme^{*}



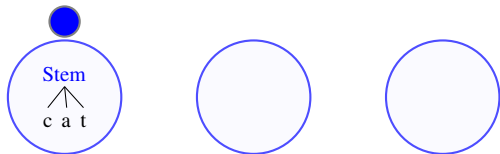
Generated words:

Adaptor grammar for stem-suffix morphology (1c)

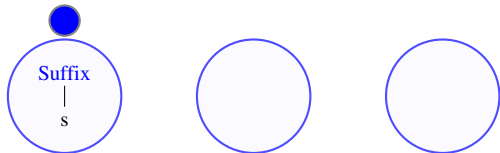
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



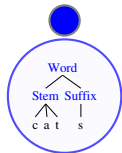
Suffix \rightarrow Phoneme^{*}



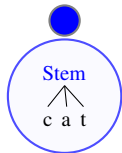
Generated words:

Adaptor grammar for stem-suffix morphology (1d)

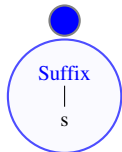
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



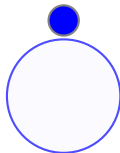
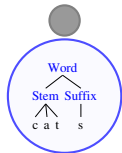
Suffix \rightarrow Phoneme^{*}



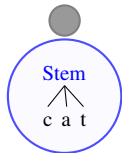
Generated words: *cats*

Adaptor grammar for stem-suffix morphology (2a)

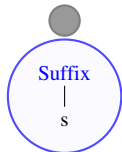
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



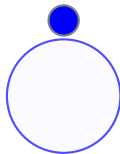
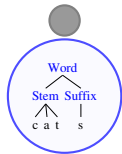
Suffix \rightarrow Phoneme^{*}



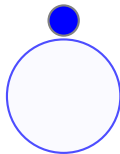
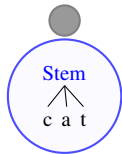
Generated words: cats

Adaptor grammar for stem-suffix morphology (2b)

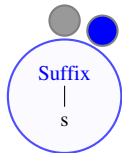
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



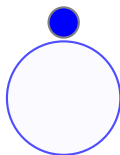
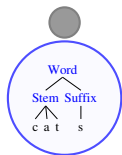
Suffix \rightarrow Phoneme^{*}



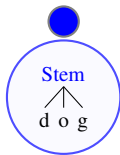
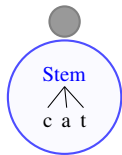
Generated words: cats

Adaptor grammar for stem-suffix morphology (2c)

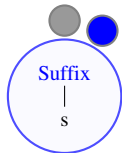
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



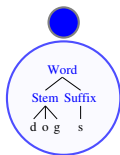
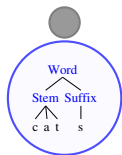
Suffix \rightarrow Phoneme^{*}



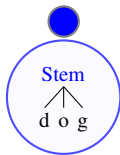
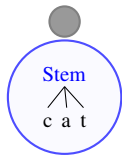
Generated words: cats

Adaptor grammar for stem-suffix morphology (2d)

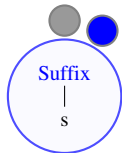
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



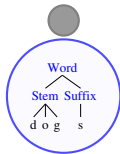
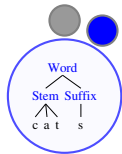
Suffix \rightarrow Phoneme^{*}



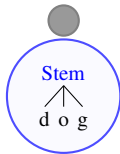
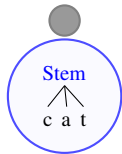
Generated words: cats, dogs

Adaptor grammar for stem-suffix morphology (3)

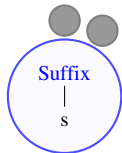
Word \rightarrow Stem Suffix



Stem \rightarrow Phoneme⁺



Suffix \rightarrow Phoneme^{*}



Generated words: cats, dogs, *cats*

Adaptor grammars as generative processes

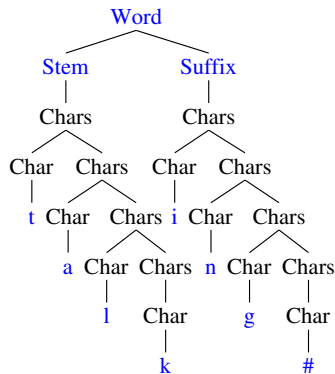
- The sequence of trees generated by an adaptor grammar are *not* independent
 - ▶ it *learns* from the trees it generates
 - ▶ if an adapted subtree has been used frequently in the past, it's more likely to be used again
- but the sequence of trees is *exchangable* (important for sampling)
- An *unadapted nonterminal* A expands using $A \rightarrow \beta$ with probability $\theta_{A \rightarrow \beta}$
- Each adapted nonterminal A is associated with a CRP (or PYP) that caches previously generated subtrees rooted in A
- An *adapted nonterminal* A expands:
 - ▶ to a subtree τ rooted in A with probability proportional to the number of times τ was previously generated
 - ▶ using $A \rightarrow \beta$ with probability proportional to $\alpha_A \theta_{A \rightarrow \beta}$

Properties of adaptor grammars

- Possible trees are generated by CFG rules
but the probability of each adapted tree is learned separately
 - Probability of adapted subtree τ is proportional to:
 - ▶ the number of times τ was seen before
⇒ “rich get richer” dynamics (Zipf distributions)
 - ▶ plus α_A times prob. of generating it via PCFG expansion
- ⇒ Useful compound structures can be *more probable than their parts*
- PCFG rule probabilities estimated *from table labels*
 - ⇒ effectively *learns from types*, not tokens
 - ⇒ makes learner less sensitive to frequency variation in input

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

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

Unsupervised word segmentation

- Input: phoneme sequences with *sentence boundaries* (Brent)
- Task: identify *word boundaries*, and hence words

y_Δu_▲w_Δa_Δn_Δt_▲t_Δu_▲s_Δi_▲D_Δ6_▲b_ΔU_Δk

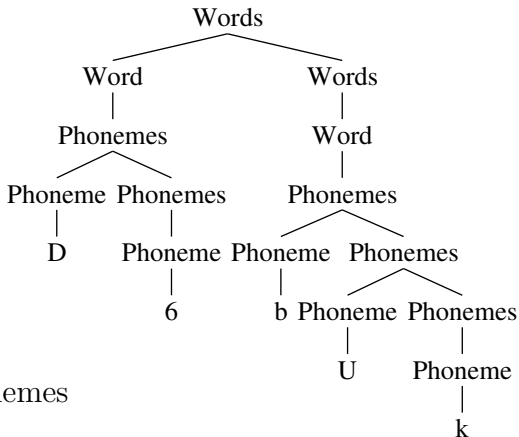
- Useful cues for word segmentation:
 - ▶ Phonotactics (Fleck)
 - ▶ Inter-word dependencies (Goldwater)

Word segmentation with PCFGs (1)

Sentence \rightarrow Word⁺
Word \rightarrow Phoneme⁺

which abbreviates

Sentence \rightarrow Words
Words \rightarrow Word Words
Word \rightarrow Phonemes
Phonemes \rightarrow Phoneme Phonemes
Phonemes \rightarrow Phoneme
Phoneme $\rightarrow a \mid \dots \mid z$

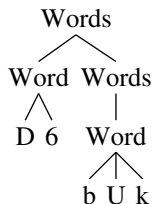


Word segmentation with PCFGs (2)

Sentence \rightarrow Word⁺

Word \rightarrow all possible phoneme strings

- But now there are an infinite number of PCFG rules!
 - ▶ once we see our (finite) training data, only finitely many are useful
- \Rightarrow the set of parameters (rules) should be chosen based on training data

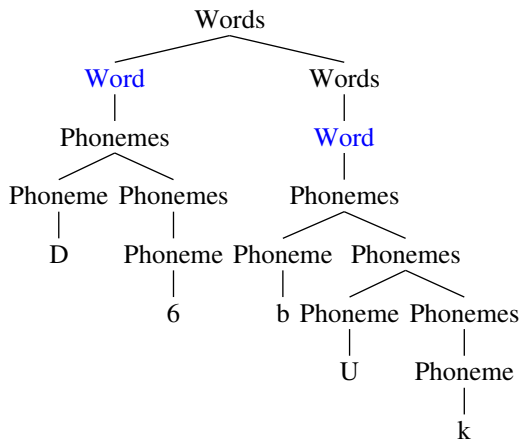


Unigram word segmentation adaptor grammar

Sentence \rightarrow Word⁺

Word \rightarrow Phoneme⁺

- *Adapted nonterminals* indicated by underlining



- Adapting Words means that the grammar learns the probability of each Word subtree independently
- Unigram word segmentation on Brent corpus: 56% token f-score

Adaptor grammar learnt from Brent corpus

- Initial grammar*

1	Sentence \rightarrow Word Sentence	1	Sentence \rightarrow Word
1	Word \rightarrow Phons		
1	Phons \rightarrow Phon Phons	1	Phons \rightarrow Phon
1	Phon \rightarrow D	1	Phon \rightarrow G
1	Phon \rightarrow A	1	Phon \rightarrow E

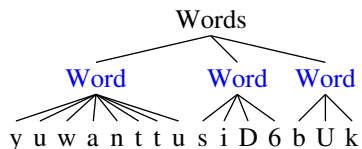
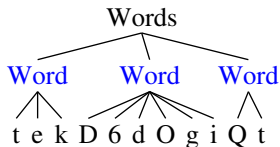
- A grammar learnt from Brent corpus*

16625	Sentence \rightarrow Word Sentence	9791	Sentence \rightarrow Word
1	Word \rightarrow Phons		
4962	Phons \rightarrow Phon Phons	1575	Phons \rightarrow Phon
134	Phon \rightarrow D	41	Phon \rightarrow G
180	Phon \rightarrow A	152	Phon \rightarrow E
460	Word \rightarrow (Phons (Phon <i>y</i>) (Phons (Phon <i>u</i>)))		
446	Word \rightarrow (Phons (Phon <i>w</i>) (Phons (Phon <i>A</i>) (Phons (Phon <i>t</i>))))		
374	Word \rightarrow (Phons (Phon <i>D</i>) (Phons (Phon <i>6</i>)))		
372	Word \rightarrow (Phons (Phon <i>&</i>) (Phons (Phon <i>n</i>) (Phons (Phon <i>d</i>))))		

Words (unigram model)

Sentence \rightarrow Word⁺ Word \rightarrow Phoneme⁺

- 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)

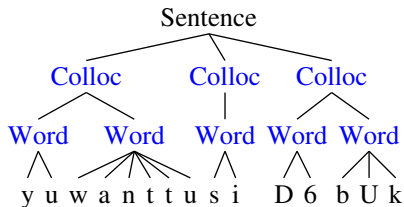


Collocations \Rightarrow Words

Sentence \rightarrow Colloc⁺

Colloc \rightarrow Word⁺

Word \rightarrow Phon⁺



- A Colloc(ation) consists of one or more words
- Both Words and Collocs are adapted (learnt)
- Significantly improves word segmentation accuracy over unigram model (76% f-score; \approx Goldwater's bigram model)

Collocations \Rightarrow Words \Rightarrow Syllables

Sentence \rightarrow Colloc⁺

Word \rightarrow Syllable

Word \rightarrow Syllable Syllable Syllable

Onset \rightarrow Consonant⁺

Nucleus \rightarrow Vowel⁺

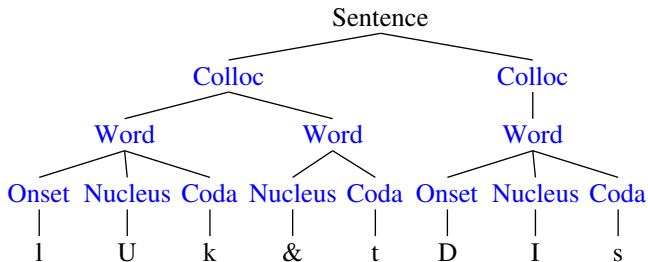
Colloc \rightarrow Word⁺

Word \rightarrow Syllable Syllable

Syllable \rightarrow (Onset) Rhyme

Rhyme \rightarrow Nucleus (Coda)

Coda \rightarrow Consonant⁺



- With no supra-word generalizations, f-score = 68%
- With 2 Collocation levels, f-score = 82%

Distinguishing internal onsets/codas helps

Sentence \rightarrow Colloc⁺

Word \rightarrow SyllableIF

Word \rightarrow SyllableI Syllable SyllableF

OnsetI \rightarrow Consonant⁺

Nucleus \rightarrow Vowel⁺

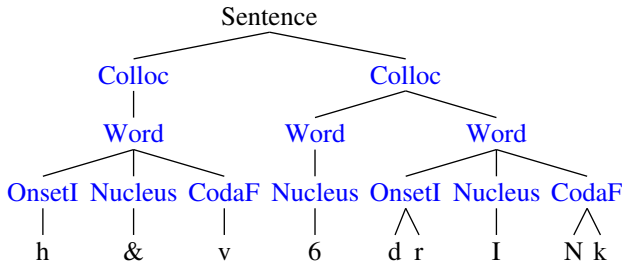
Colloc \rightarrow Word⁺

Word \rightarrow SyllableI SyllableF

SyllableIF \rightarrow (OnsetI) RhymeF

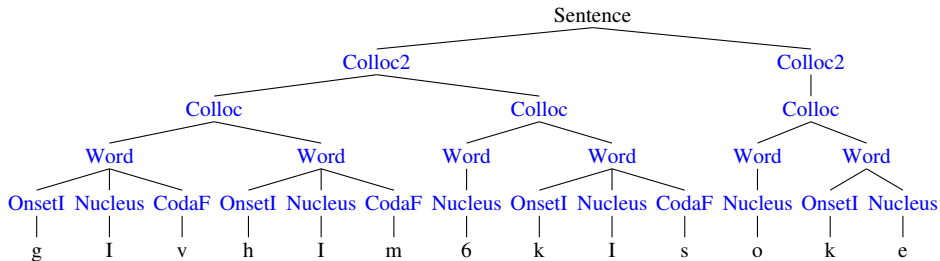
RhymeF \rightarrow Nucleus (CodaF)

CodaF \rightarrow Consonant⁺



- Without distinguishing initial/final clusters, f-score = 82%
- Distinguishing initial/final clusters, f-score = 84%
- With 2 Collocation levels, f-score = 87%

Collocations² ⇒ Words ⇒ Syllables

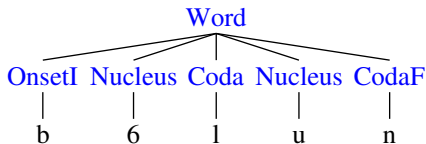


Syllabification learnt by adaptor grammars

- Grammar has no reason to prefer to parse word-internal intervocalic consonants as onsets

1 Syllable \rightarrow Onset Rhyme 1 Syllable \rightarrow Rhyme

- The learned grammars consistently analyse them as either Onsets or Codas \Rightarrow learns wrong grammar half the time

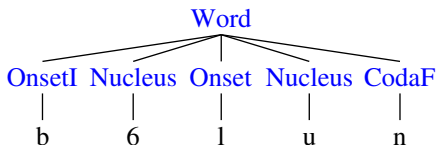


- Syllabification accuracy is relatively poor
Syllabification given true word boundaries: f-score = 83%
Syllabification learning word boundaries: f-score = 74%

Preferring Onsets improves syllabification

2 Syllable \rightarrow Onset Rhyme 1 Syllable \rightarrow Rhyme

- Changing the prior to prefer word-internal Syllables with Onsets dramatically improves segmentation accuracy
- “Rich get richer” property of Chinese Restaurant Processes
 \Rightarrow all ambiguous word-internal consonants analysed as Onsets



- Syllabification accuracy is much higher than without bias
Syllabification given true word boundaries: f-score = 97%
Syllabification learning word boundaries: f-score = 90%

Modelling sonority classes improves syllabification

$\text{Onset} \rightarrow \text{Onset}_{\text{Stop}}$	$\text{Onset} \rightarrow \text{Onset}_{\text{Fricative}}$
$\text{Onset}_{\text{Stop}} \rightarrow \text{Stop}$	$\text{Onset}_{\text{Stop}} \rightarrow \text{Stop Onset}_{\text{Fricative}}$
$\text{Stop} \rightarrow \text{p}$	$\text{Stop} \rightarrow \text{t}$

- Five consonant sonority classes
- $\text{Onset}_{\text{Stop}}$ generates a consonant cluster with a Stop at left edge
- Prior prefers transitions compatible with sonority hierarchy (e.g., $\text{Onset}_{\text{Stop}} \rightarrow \text{Stop Onset}_{\text{Fricative}}$) to transitions that aren't (e.g., $\text{Onset}_{\text{Fricative}} \rightarrow \text{Fricative Onset}_{\text{Stop}}$)
- Same transitional probabilities used for initial and non-initial Onsets (maybe not a good idea for English?)
- Word-internal Onset bias still necessary
- Syllabification given true boundaries: f-score = 97.5%
- Syllabification learning word boundaries: f-score = 91%

Summary: Adaptor grammars for word segmentation

- Easy to define adaptor grammars that are sensitive to:

Generalization	Accuracy
words as units (unigram)	56%
+ associations between words (collocations)	76%
+ syllable structure	87%

- word segmentation *improves when you learn other things as well*
 - ▶ *explain away* potentially misleading generalizations

Another application of adaptor grammars: Learning structure in names

- Many different kinds of names
 - ▶ Person names, e.g., *Mr. Sam Spade Jr.*
 - ▶ Company names, e.g., *United Motor Manufacturing Corp.*
 - ▶ Other names, e.g., *United States of America*
- At least some of these are structured; e.g., *Mr* is an honorific, *Sam* is first name, *Spade* is a surname, etc.
- Penn treebanks assign flat structures to base NPs (including names)
- Data set: 10,787 unique lowercased sequences of base NP proper nouns, containing 23,392 words
- Can we automatically learn the structure of these names?

Adaptor grammar for names

NP \rightarrow Unordered⁺

NP \rightarrow (A0) (A1) ... (A6)

A0 \rightarrow Word⁺

...

A6 \rightarrow Word⁺

Unordered \rightarrow Word⁺

NP \rightarrow (B0) (B1) ... (B6)

B0 \rightarrow Word⁺

...

B6 \rightarrow Word⁺

- *Sample output:*

(A0 barrett) (A3 smith)

(A0 albert) (A2 j.) (A3 smith) (A4 jr.)

(A0 robert) (A2 b.) (A3 van dover)

(B0 aim) (B1 prime rate) (B2 plus) (B5 fund) (B6 inc.)

(B0 balfour) (B1 maclaine) (B5 international) (B6 ltd.)

(B0 american express) (B1 information services) (B6 co)

(U abc) (U sports)

(U sports illustrated)

(U sports unlimited)

Outline

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

What do we have to learn?

- To learn an adaptor grammar, we need:
 - ▶ probabilities of grammar rules
 - ▶ adapted subtrees and their probabilities for adapted non-terminals
- If we knew the true parse trees for a training corpus, we could:
 - ▶ read off the adapted subtrees from the corpus
 - ▶ count rules and adapted subtrees in corpus
 - ▶ compute the rule and subtree probabilities from these counts
 - simple computation (smoothed relative frequencies)
- If we aren't given the parse trees:
 - ▶ there are usually *infinitely many* possible adapted subtrees
 - ⇒ can't track the probability of all of them (as in EM)
 - ▶ but *sample parses of a finite corpus* only include finitely many
- Sampling-based methods learn the relevant subtrees as well as their weights

If we had infinite data . . .

- A simple incremental learning algorithm:
 - ▶ Repeat forever:
 - get next sentence
 - sample a parse tree for sentence according to current grammar
 - increment rule and adapted subtree counts with counts from sampled parse tree
 - update grammar according to these counts
- *Particle filter* learners update *multiple versions of the grammar* at each sentence

A Gibbs sampler for learning adaptor grammars

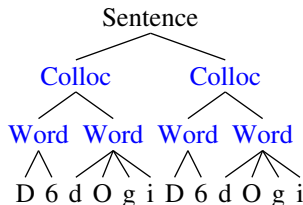
- Intuition: same as simple incremental algorithm, but re-use sentences in training data
 - ▶ Assign (random) parse trees to each sentence, and compute rule and subtree counts
 - ▶ Repeat forever:
 - pick a sentence (and corresponding parse) at random
 - deduct the counts for the sentence's parse from current rule and subtree counts
 - sample a parse for sentence according to updated grammar
 - add sampled parse's counts to rule and subtree counts
- Sampled parse trees and grammar converges to Bayesian posterior distribution

Sampling parses from an adaptor grammar

- Sampling a parse tree for a sentence is computationally most demanding part of learning algorithm
- Component-wise Metropolis-within-Gibbs sampler for parse trees:
 - ▶ adaptor grammar rules and probabilities *change on the fly*
 - ▶ construct PCFG *proposal grammar* from adaptor grammar for previous sentences
 - ▶ sample a parse from PCFG proposal grammar
 - ▶ use accept/reject to convert samples from proposal PCFG to samples from adaptor grammar
- For particular adaptor grammars, there are often more efficient algorithms

Details about sampling parses

- Adaptor grammars are *not context-free*
- The probability of a rule (and a subtree) can change within a single sentence
 - ▶ breaks standard dynamic programming
- But with moderate or large corpora, the probabilities don't change by much
 - ▶ use Metropolis-Hastings accept/reject with a PCFG proposal distribution
- Rules of PCFG proposal grammar $G'(\mathbf{t}_{-j})$ 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 τ in A 's restaurant:
 $\theta'_{A \rightarrow \text{YIELD}(\tau)} \propto n_\tau$, the number of customers at table τ
- Parses of $G'(\mathbf{t}_{-j})$ can be mapped back to adaptor grammar parses



Summary: learning adaptor grammars

- *Naive integrated parsing/learning algorithm*:
 - ▶ *sample* a parse for next sentence
 - ▶ *count* how often each adapted structure appears in parse
- Sampling parses addresses *exploration/exploitation dilemma*
- First few sentences receive random segmentations
⇒ this algorithm does *not* optimally learn from data
- *Gibbs sampler* batch learning algorithm
 - ▶ assign every sentence a (random) parse
 - ▶ repeatedly cycle through training sentences:
 - withdraw parse (decrement counts) for sentence
 - sample parse for current sentence and update counts
- *Particle filter* online learning algorithm
 - ▶ Learn different versions (“particles”) of grammar at once
 - ▶ For each particle sample a parse of next sentence
 - ▶ Keep/replicate particles with high probability parses

Outline

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

Summary and future work

- Adaptor Grammars (AG) “adapt” to the strings they generate
- AGs learn probability of whole subtrees (not just rules)
- AGs are *non-parametric* because cached subtrees depend on the data
- AGs inherit the “rich get richer” property from Chinese Restaurant Processes
 - ⇒ AGs generate Zipfian distributions
 - ⇒ learning is driven by types rather than tokens
- AGs can be used to describe a variety of linguistic inference problems
- Sampling methods are a natural approach to AG inference

Outline

A Primer on Bayesian inference

Probabilistic Context-Free Grammars

Chinese Restaurant Processes and Nonparametric Bayes

Adaptor grammars

Adaptor grammars for unsupervised word segmentation

Bayesian inference for adaptor grammars

Conclusion

Extending Adaptor Grammars

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 \mathcal{T}_A of trees.

These are the smallest sets satisfying:

- If $A \in W$ then $\mathcal{T}_A = \{A\}$.
- If $A \in N$ then:

$$\mathcal{T}_A = \bigcup_{A \rightarrow B_1 \dots B_n \in R_A} \text{TREE}_A(\mathcal{T}_{B_1}, \dots, \mathcal{T}_{B_n})$$

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

$$\text{TREE}_A(\mathcal{T}_{B_1}, \dots, \mathcal{T}_{B_n}) = \left\{ \begin{array}{l} A \\ \wedge \\ t_1 \dots t_n \end{array} : \begin{array}{l} t_i \in \mathcal{T}_{B_i}, \\ i = 1, \dots, n \end{array} \right\}$$

The set of trees generated by a CFG is \mathcal{T}_S .

Probabilistic context-free grammars

A *probabilistic context-free grammar* (PCFG) is a CFG and a vector θ , 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 \mathcal{T}_A for $A \in N \cup W$:

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

where δ_A puts all its mass onto the singleton tree A , and:

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

$\text{TD}_A(G_1, \dots, G_n)$ is a distribution over \mathcal{T}_A where each subtree t_i is generated independently from G_i .

DP adaptor grammars

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

$$\begin{aligned} G_A &\sim \text{DP}(\alpha_A, H_A) \text{ if } \alpha_A > 0 \\ &= H_A \text{ if } \alpha_A = 0 \end{aligned}$$

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

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 (\mathbf{G}, \mathbf{H}) is defined by:

$$\begin{aligned} G_A &\sim \text{DP}(\alpha_A, H_A) \text{ if } \alpha_A > 0 \\ &= H_A \quad \quad \quad \text{if } \alpha_A = 0 \end{aligned}$$

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

- This holds *even if adaptor grammar is recursive*
- Question: when does this define a *distribution* over (\mathbf{G}, \mathbf{H}) ?