Inference for PCFGs and Adaptor Grammars

NIPS grammar induction workshop

Mark Johnson

Macquarie University
Sydney, Australia

December 2009

Talk outline

- Bayesian inference for PCFG rule probabilities
  - collapsed Gibbs sampler (integrates out rule probabilities)
- Learn grammar rules by *non-parametric Bayes*
  $\Rightarrow$ *adaptor grammars*
  - there are an *unbounded number of potential rules*
  - but a finite number of finite parses can only use a finite number of rules

  $\Rightarrow$ only explicitly represent the rules used in previous parse samples
- Improvements to adaptor grammars and MCMC inference procedure
  - *estimating hyper-parameters* using slice sampling
  - *modal decoding* from multiple samples from multiple runs
  - *random initialization* instead of incremental initialization
  - *table label resampling* as well as sentence resampling

Outline

Inference for PCFG rule probabilities

Learning grammar rules (not just probabilities)

Adaptor grammars

Non-parametric inference in word segmentation
- Priors on Adaptor Grammar PYP parameters

Unsupervised inference of Adaptor Grammars
- Modal word segmentation
- Random vs incremental initialization
- Table label resampling

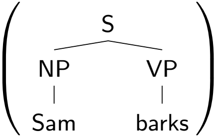
Conclusion

Probabilistic context-free grammars

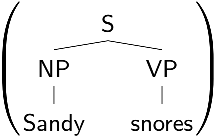
- Rules in *Context-Free Grammars* (CFGs) expand nonterminals into sequences of terminals and nonterminals
- A *Probabilistic CFG* (PCFG) associates each nonterminal with a multinomial distribution over the rules that expand it
- Probability of a tree is the *product of the probabilities of the rules*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

Rule r	θ_r
S $\longrightarrow$ NP VP	1.0
NP $\longrightarrow$ Sam	0.75
VP $\longrightarrow$ barks	0.6

Rule r	θ_r
NP $\longrightarrow$ Sandy	0.25
VP $\longrightarrow$ snores	0.4

P 

$= 0.45$

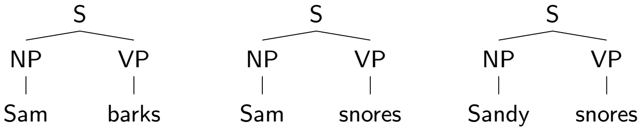
P 

$= 0.1$

Maximum likelihood estimation from visible parses

- Each rule expansion is sampled from parent's multinomial

$\Rightarrow$ Maximum Likelihood Estimator (MLE) is rule's *relative frequency*



Rule r	n_r	θ_r
S $\rightarrow$ NP VP	3	1.0
NP $\rightarrow$ Sam	2	0.66
VP $\rightarrow$ barks	1	0.33

Rule r	n_r	θ_r
NP $\rightarrow$ Sandy	1	0.33
VP $\rightarrow$ snores	2	0.66

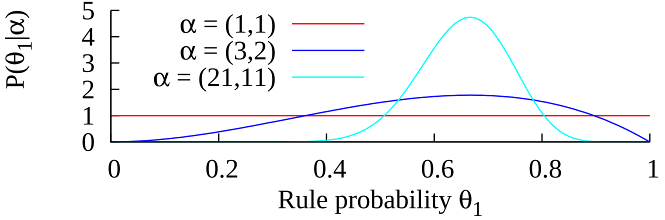
- But MLE is often *overly certain*, especially with sparse data
  - E.g., "accidental zeros" $n_r = 0 \Rightarrow \theta_r = 0$.

Bayesian estimation from visible parses

- Bayesian estimators estimate a *distribution* over rule probabilities

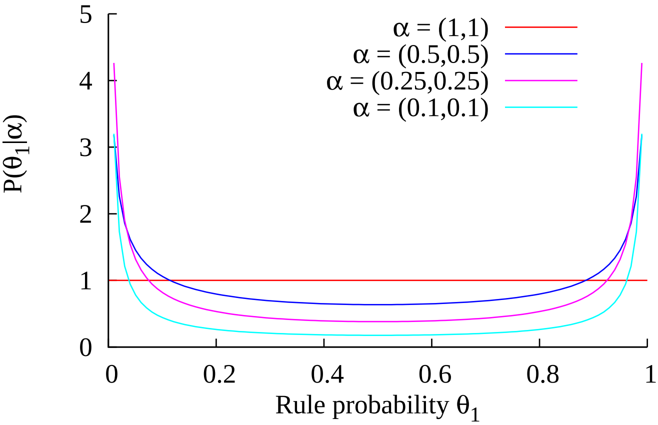
$$\underbrace{\mathrm{P}(\boldsymbol{\theta} \mid \boldsymbol{n})}_{\text{Posterior}} \propto \underbrace{\mathrm{P}(\boldsymbol{n} \mid \boldsymbol{\theta})}_{\text{Likelihood}} \underbrace{\mathrm{P}(\boldsymbol{\theta})}_{\text{Prior}}$$

- *Dirichlet distributions* are conjugate priors for multinomials
  - A Dirichlet distribution over $(\theta_1, \ldots, \theta_m)$ is specified by positive parameters $(\alpha_1, \ldots, \alpha_m)$
  - If Prior $= \mathrm{Dir}(\boldsymbol{\alpha})$ then Posterior $= \mathrm{Dir}(\boldsymbol{\alpha} + \boldsymbol{n})$



Sparse Dirichlet priors

- As $\alpha \to 0$, Dirichlet distributions become peaked around 0
  "Grammar includes some of these rules, but we don't know which!"



Estimating rule probabilities from strings alone

- Input: *terminal strings* and *grammar rules*
- Output: *rule probabilities* $\boldsymbol{\theta}$
- In general, no closed-form solution for $\boldsymbol{\theta}$
  - iterative algorithms usually involving *repeatedly reparsing training data*
- *Expectation Maximization* (EM) procedure generalizes visible data ML estimators to hidden data problems
- *Inside-Outside algorithm* is a cubic-time EM algorithm for PCFGs
- Bayesian estimation of $\boldsymbol{\theta}$ via:
  - *Variational Bayes* or
  - *Markov Chain Monte Carlo* (MCMC) methods such as *Gibbs sampling*

Gibbs sampler for parse trees and rule probabilities

- Input: *terminal strings* $(x_1, \ldots, x_n)$, *grammar rules* and *Dirichlet prior parameters* $\boldsymbol{\alpha}$
- Output: stream of sample *rule probabilities* $\boldsymbol{\theta}$ and *parse trees* $\boldsymbol{t} = (t_1, \ldots, t_n)$
- Algorithm:

  Assign parse trees to the strings somehow (e.g., randomly)
  Repeat forever:
      Compute rule counts $\boldsymbol{n}$ from $\boldsymbol{t}$
      Sample $\boldsymbol{\theta}$ from $\mathrm{Dir}(\boldsymbol{\alpha} + \boldsymbol{n})$
      For each string x_i:
          replace t_i with a tree sampled from $\mathrm{P}(t | x_i, \boldsymbol{\theta})$.

- After *burn-in*, $(\boldsymbol{\theta}, \boldsymbol{t})$ are distributed according to Bayesian posterior
- Sampling parse tree from $\mathrm{P}(t | x_i, \boldsymbol{\theta})$ involves parsing string x_i.

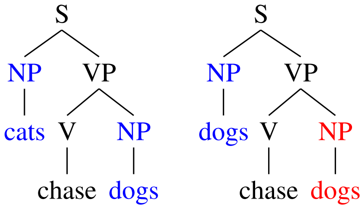
Collapsed Gibbs samplers

- *Integrate out* rule probabilities $\boldsymbol{\theta}$ to obtain *predictive distribution* $\mathrm{P}(t_i|x_i, \boldsymbol{t}_{-i})$ of parse t_i for sentence x_i given other parses $\boldsymbol{t}_{-i}$
- *Collapsed Gibbs sampler*

  For each sentence x_i in training data:

  Replace t_i with a sample from $\mathrm{P}(t|x_i, \boldsymbol{t}_{-i})$
- A problem: $\mathrm{P}(t_i|x_i, t_{-i})$ is *not a PCFG distribution*

  $\Rightarrow$ no dynamic-programming sampler (AFAIK)



Metropolis-Hastings samplers

- Metropolis-Hastings (MH) acceptance-rejection procedure uses samples from a *proposal distribution* to produce samples from a *target distribution*
- When sentence size $\ll$ training data size, $\mathrm{P}(t_i|x_i, \boldsymbol{t}_{-i})$ is almost a PCFG distribution
  - use a PCFG approximation based on $\boldsymbol{t}_{-i}$ as *proposal distribution*
  - apply MH to transform proposals to $\mathrm{P}(t_i|x_i, \boldsymbol{t}_{-i})$
- To construct a Metropolis-Hastings sampler you need to be able to:
  - efficiently sample from proposal distribution
  - calculate ratios of parse probabilities under proposal distribution
  - calculate ratios of parse probabilities under target distribution

Collapsed Metropolis-within-Gibbs sampler for PCFGs

- Input: *terminal strings* $(x_1, \ldots, x_n)$, *grammar rules* and *Dirichlet prior parameters* $\boldsymbol{\alpha}$
- Output: stream of sample *parse trees* $\boldsymbol{t} = (t_1, \ldots, t_n)$
- Algorithm:

  Assign parse trees to the strings somehow (e.g., randomly)
  Repeat forever:
    For each sentence x_i in training data:
      Compute rule counts $\boldsymbol{n}_{-i}$ from $\boldsymbol{t}_{-i}$
      Compute proposal grammar probabilities $\boldsymbol{\theta}$ from $\boldsymbol{n}_{-i}$
      Sample a tree t from $\mathrm{P}(t|x_i, \boldsymbol{\theta})$
      Replace t_i with t according to
          Metropolis-Hastings accept-reject formula

Q: Are there efficient *local-move* tree samplers?

- DP PCFG samplers require $O(n^3)$ time per sample
  - are there faster samplers?
- For HMMs, a *point-wise* sampler resamples the state labeling *one word* at a time
  - All state *sequences* reachable by one or more such moves
  - Transition probabilities don't require dynamic programming
    $\Rightarrow$ no need for MH
- Question: *Are there efficiently-computable local moves that can transform any parse into any other parse of same string?*
  - for HMMs, DP sampler generally more effective than point-wise sampler
  - point-wise samplers could be more effective for grammars with complex DP parsing algorithms (e.g., TAG, CCG, LFG, HPSG)

Outline

Inference for PCFG rule probabilities

Learning grammar rules (not just probabilities)

Adaptor grammars

Non-parametric inference in word segmentation
- Priors on Adaptor Grammar PYP parameters

Unsupervised inference of Adaptor Grammars
- Modal word segmentation
- Random vs incremental initialization
- Table label resampling

Conclusion

Learning grammar rules

- Input: *terminal strings*
- Output: *grammar rules* and *rule probabilities* $\boldsymbol{\theta}$
- "Generate and test" approach (Carroll and Charniak, Stolcke)

  Guess an initial set of rules

  Repeat:

  re-estimate rule probabilities from strings

  prune low probability rules

  propose additional potentially useful rules
- Non-parametric Bayesian methods seem to provide a more systematic approach

Non-parameteric Bayesian extensions to PCFGs

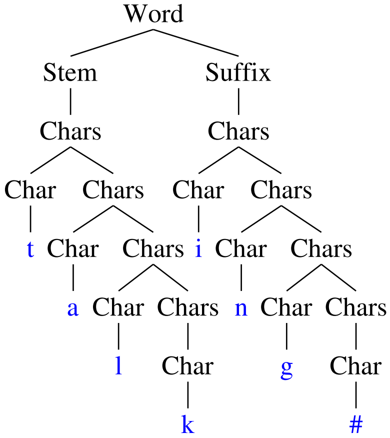
- Non-parametric $\Rightarrow$ no fixed set of parameters
- Two obvious non-parametric extensions to PCFGs:
  - let the set of non-terminals grow unboundedly
    - given an initial grammar with coarse-grained categories, split non-terminals into more refined categories $\mathrm{S}_{12} \rightarrow \mathrm{NP}_7\,\mathrm{VP}_4$ instead of $\mathrm{S} \rightarrow \mathrm{NP}\,\mathrm{VP}$.
    - PCFG generalization of "infinite HMM".
  - let the set of rules grow unboundedly $\Rightarrow$ *adaptor grammars*
    - use a (meta-)grammar to generate potential rules
    - learn subtrees and their probabilities
      i.e., tree substitution grammar, where we learn the fragments as well as their probabilities
- No reason both can't be done at once ...

Learning syntax is hard!

- Can formulate learning syntax as Bayesian estimation
- On toy data, Bayesian estimators do well
- Results are disappointing on "real" data
  - wrong algorithm?
  - wrong kind of grammar?
  - wrong type of training data?
- This paper focuses on learning grammars for simpler phenomena:
  - Morphological segmentation (e.g., *walking* = *walk*+*ing*)
  - Word segmentation of unsegmented phoneme sequences
  - Learning collocations in topic models
  - Learning internal structure of named-entity NPs

A CFG for stem-suffix morphology

$$
\begin{aligned}
\text{Word} &\rightarrow \text{Stem Suffix} & \qquad \text{Chars} &\rightarrow \text{Char} \\
\text{Stem} &\rightarrow \text{Chars} & \qquad \text{Chars} &\rightarrow \text{Char Chars} \\
\text{Suffix} &\rightarrow \text{Chars} & \qquad \text{Char} &\rightarrow \text{a} \mid \text{b} \mid \text{c} \mid \ldots
\end{aligned}
$$



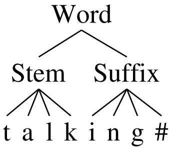
- Grammar's trees can represent any segmentation of words into stems and suffixes

$\Rightarrow$ 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

$$
\begin{array}{rcl}
\text{Word} & \rightarrow & \text{Stem Suffix} \\
\text{Stem} & \rightarrow & \textit{all possible stems} \\
\text{Suffix} & \rightarrow & \textit{all possible suffixes}
\end{array}
$$



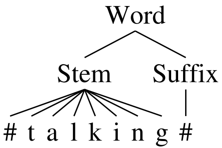
- A rule for each morpheme
  $\Rightarrow$ "PCFG" can represent probability of each morpheme
- *Unbounded number of possible rules, so this is not a PCFG*
  - not a practical problem, as only a finite set of rules could possibly be used in any particular data set

Maximum likelihood estimate for $\boldsymbol{\theta}$ is trivial

- Maximum likelihood selects $\boldsymbol{\theta}$ that minimizes KL-divergence between model and training data $\boldsymbol{W}$ distributions
- *Saturated model* in which each word is generated by its own rule replicates training data distribution $\boldsymbol{W}$ exactly

⇒ Saturated model is maximum likelihood estimate

- Maximum likelihood estimate does not find any suffixes



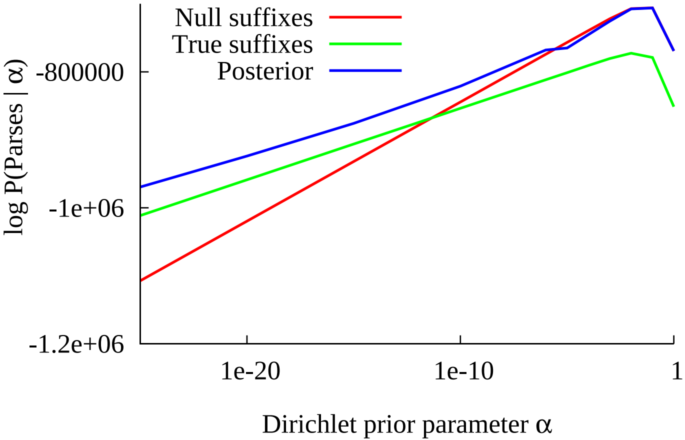
Forcing generalization using Dirichlet priors

- Maximum Likelihood solution analyses each word as a separate stem
  - fails to generalize
  - one non-zero probability rule per word type in data
- Dirichlet prior prefers $\theta = 0$ when $\alpha \to 0$
  - use Dirichlet prior to prefer *sparse rule probability vectors*
- Following experiments use orthographic verbs from U Penn. WSJ treebank

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

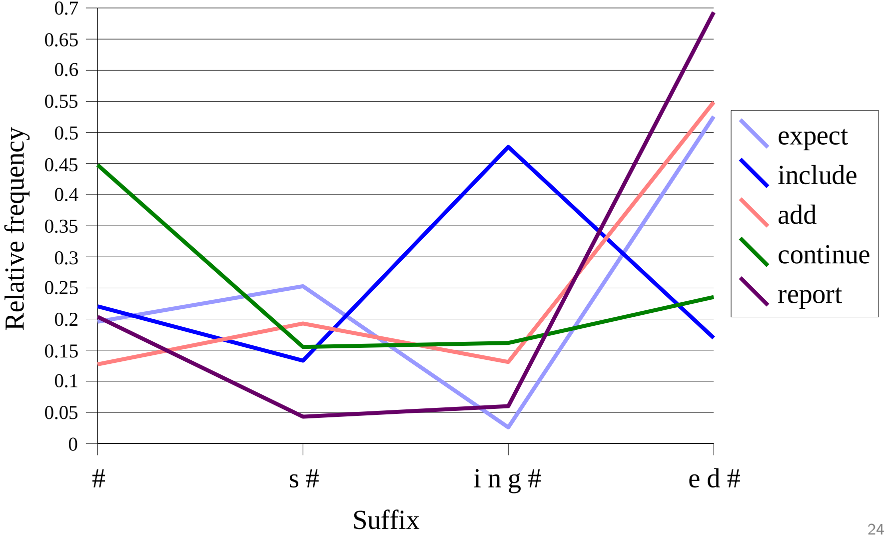


- Correct solution is nowhere near as likely as posterior

$\Rightarrow$ model is wrong!

Inflection is not statistically independent

Word $\longrightarrow$ Stem Suffix



Types and tokens

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

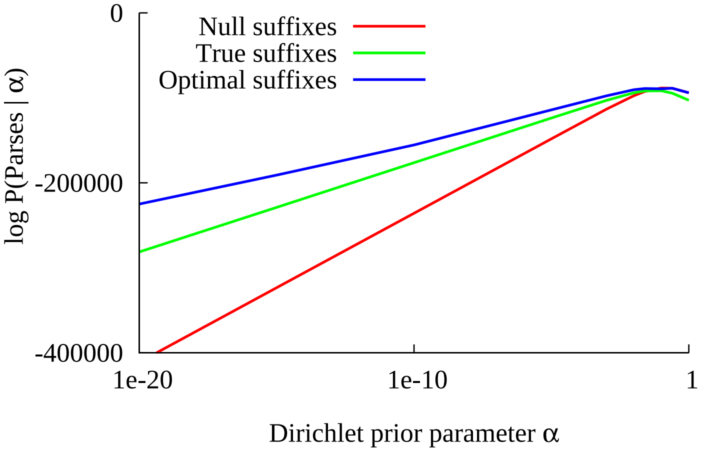
$$\begin{aligned} \text{Data} &= \text{“the cat chased the other cat”} \\ \text{Tokens} &= \text{“the”, “cat”, “chased”, “the”, “other”, “cat”} \\ \text{Types} &= \text{“the”, “cat”, “chased”, “other”} \end{aligned}$$

- Estimating θ from *word types* rather than word tokens eliminates (most) frequency variation
  - 4 common verb suffixes, so when estimating from verb types $\theta_{\mathsf{Suffix}\rightarrow\mathtt{i\,n\,g\,\#}} \approx 0.25$
- 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*

$\alpha = 0.1$		$\alpha = 10^{-5}$		$\alpha = 10^{-10}$		$\alpha = 10^{-15}$	
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
  $\Rightarrow$ generalize over groups of rules
  $\Rightarrow$ units of generalization should be chosen based on data
- Type-based inference mitigates over-dispersion
  $\Rightarrow$ 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 Sharon Goldwater's models

Outline

Inference for PCFG rule probabilities

Learning grammar rules (not just probabilities)

Adaptor grammars

Non-parametric inference in word segmentation
  Priors on Adaptor Grammar PYP parameters

Unsupervised inference of Adaptor Grammars
  Modal word segmentation
  Random vs incremental initialization
  Table label resampling

Conclusion

Adaptor grammars: informal description

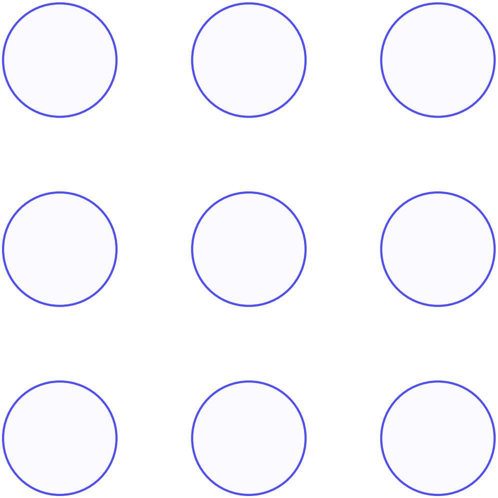
- The trees generated by an adaptor grammar are defined by CFG rules as in a CFG
- A subset of the nonterminals are *adapted*
  - each adapted nonterminal A has a *concentration parameter* α_A
- An *unadapted nonterminal* A expands using $A \rightarrow \beta$ with probability $\theta_{A\rightarrow\beta}$
- 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 $\alpha_A \theta_{A\rightarrow\beta}$

Adaptor grammar for stem-suffix morphology (0)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



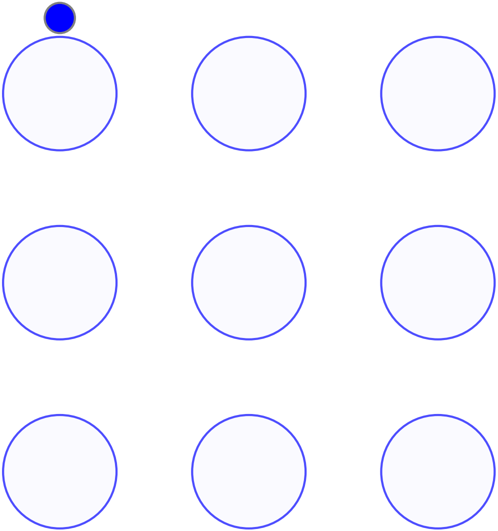
Generated words:

Adaptor grammar for stem-suffix morphology (1a)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



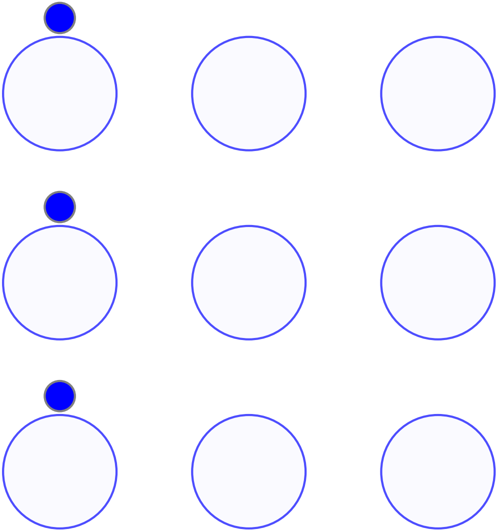
Generated words:

Adaptor grammar for stem-suffix morphology (1b)

<u>Word</u> $\rightarrow$ Stem Suffix

<u>Stem</u> $\rightarrow$ Phoneme^{+}

<u>Suffix</u> $\rightarrow$ Phoneme*



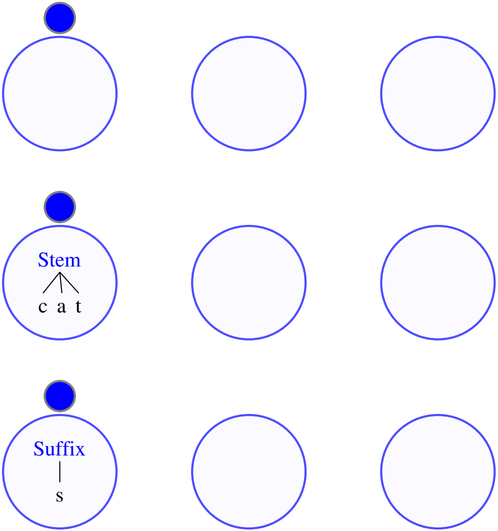
Generated words:

Adaptor grammar for stem-suffix morphology (1c)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



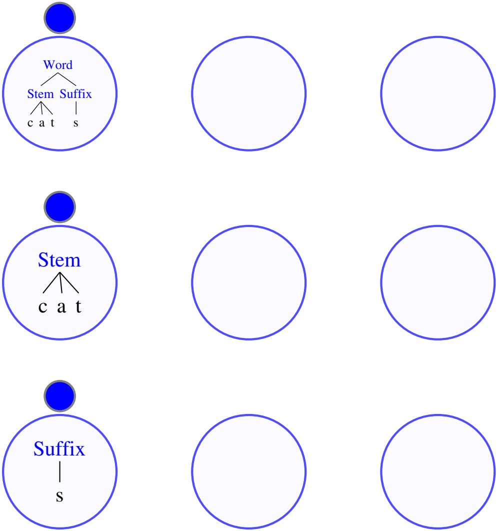
Generated words:

Adaptor grammar for stem-suffix morphology (1d)

Word $\longrightarrow$ Stem Suffix

Stem $\longrightarrow$ Phoneme^{+}

Suffix $\longrightarrow$ Phoneme*



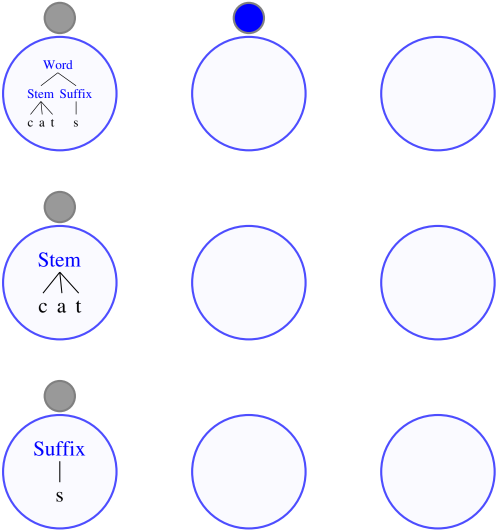
Generated words: cats

Adaptor grammar for stem-suffix morphology (2a)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



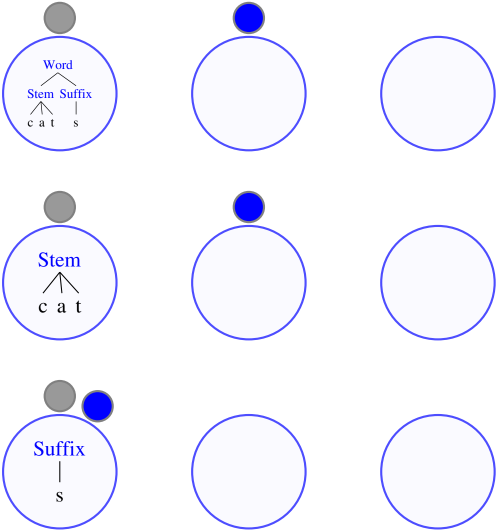
Generated words: cats

Adaptor grammar for stem-suffix morphology (2b)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



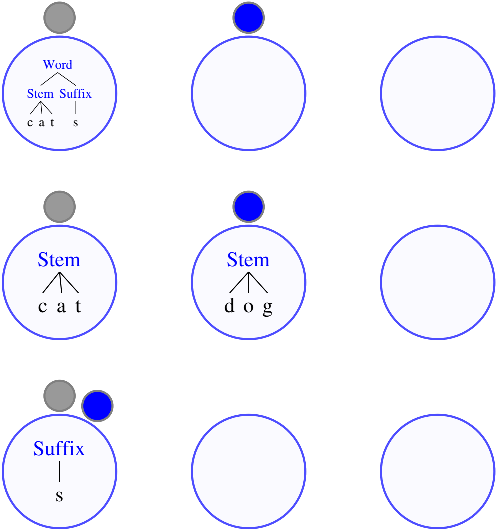
Generated words: cats

Adaptor grammar for stem-suffix morphology (2c)

Word → Stem Suffix

Stem → Phoneme^{+}

Suffix → Phoneme*



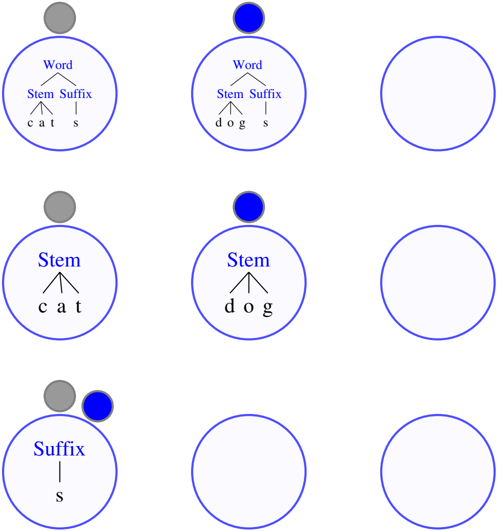
Generated words: cats

Adaptor grammar for stem-suffix morphology (2d)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



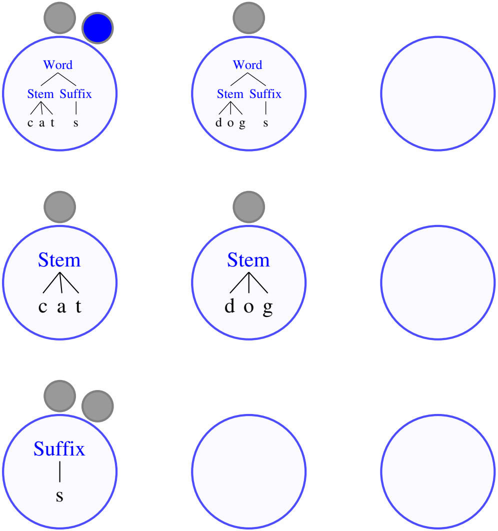
Generated words: cats, dogs

Adaptor grammar for stem-suffix morphology (3)

Word → Stem Suffix

Stem → Phoneme^{+}

Suffix → Phoneme*



Generated words: cats, dogs, cats

Adaptor grammars as generative processes

- *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 nonterminal A has a CRP (or PYP) that caches previously generated subtrees rooted in A
- The CFG rules of the adapted nonterminals determine the *base distributions* of these CRPs/PYPs
- The trees generated by an adaptor grammar are *not* independent
  - if an adapted subtree has been used frequently in the past, it's more likely to be used again

  $\Rightarrow$ an adaptor grammar *learns* from the trees it generates
- but the sequence of trees is *exchangable* (important for sampling)

Properties of adaptor grammars

- Possible trees generated by CFG rules
  but the probability of each adapted tree is estimated separately
- Probability of adapted nonterminal A expanding to subtree τ is proportional to:
  - the number of times τ was seen before
    $\Rightarrow$ "rich get richer" dynamics (Zipf distributions)
  - plus α_A times prob. of generating it via PCFG expansion
    $\Rightarrow$ nonzero but decreasing probability of novel structures

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

- Base PCFG rule probabilities estimated *from table labels*
  $\Rightarrow$ learns from types, not tokens

Outline

Inference for PCFG rule probabilities

Learning grammar rules (not just probabilities)

Adaptor grammars

Non-parametric inference in word segmentation
- Priors on Adaptor Grammar PYP parameters

Unsupervised inference of Adaptor Grammars
- Modal word segmentation
- Random vs incremental initialization
- Table label resampling

Conclusion

Unsupervised word segmentation

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

$$\mathrm{y}_{\vartriangle}\mathrm{u}_{\blacktriangle}\mathrm{w}_{\vartriangle}\mathrm{a}_{\vartriangle}\mathrm{n}_{\vartriangle}\mathrm{t}_{\blacktriangle}\mathrm{t}_{\vartriangle}\mathrm{u}_{\blacktriangle}\mathrm{s}_{\vartriangle}\mathrm{i}_{\blacktriangle}\mathrm{D}_{\vartriangle}6_{\blacktriangle}\mathrm{b}_{\vartriangle}\mathrm{U}_{\vartriangle}\mathrm{k}$$

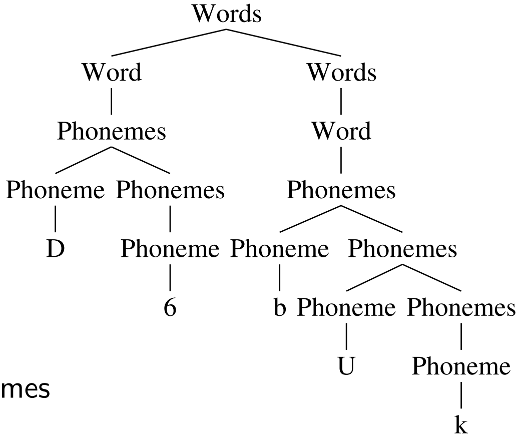
- Useful cues for word segmentation:
  - Phonotactics (Fleck)
  - Inter-word dependencies (Goldwater)

Word segmentation with PCFGs (1)

$$\text{Sentence} \rightarrow \text{Word}^{+}$$
$$\text{Word} \rightarrow \text{Phoneme}^{+}$$

which abbreviates

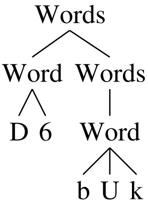
Sentence $\rightarrow$ Words
Words $\rightarrow$ Word Words
Word $\rightarrow$ Phonemes
Phonemes $\rightarrow$ Phoneme Phonemes
Phonemes $\rightarrow$ Phoneme
Phoneme $\rightarrow$ $a \mid \ldots \mid z$



Word segmentation with PCFGs (1)

Sentence $\rightarrow$ Word^{+}
Word $\rightarrow$ all possible phoneme strings

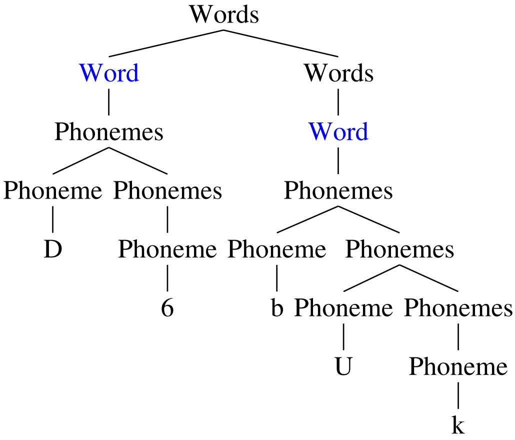
- But now there are an infinite number of PCFG rules!
  - once we see our (finite) training data, only finitely many are useful
  - $\Rightarrow$ the set of parameters (rules) should be chosen based on training data



Unigram word segmentation adaptor grammar

Sentence → Word$^+$
Word → 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

Unigram adaptor grammar after learning

- Given the Brent corpus and the unigram adaptor grammar

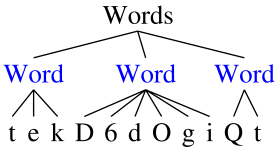
$$\text{Words} \rightarrow \text{Word}^{+}$$
$$\underline{\text{Word}} \rightarrow \text{Phon}^{+}$$

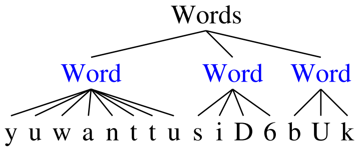
the learnt adapted grammar contains 1,712 rules such as:

15758	Words $\rightarrow$ Word Words
9791	Words $\rightarrow$ Word
1660	Word $\rightarrow$ Phon^{+}
402	Word $\rightarrow$ $y\ u$
137	Word $\rightarrow$ $I\ n$
111	Word $\rightarrow$ $w\ I\ T$
100	Word $\rightarrow$ $D\ 6\ d\ O\ g\ i$
45	Word $\rightarrow$ $I\ n\ D\ 6$
20	Word $\rightarrow$ $I\ n\ D\ 6\ h\ Q\ s$

unigram: Words

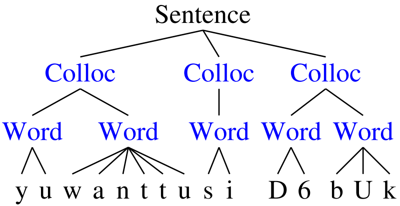
- Unigram word segmentation model assumes each word is generated independently
- But there are strong inter-word dependencies (collocations)
- Unigram model can only capture such dependencies by analyzing collocations as words (Goldwater 2006)





colloc: Collocations ⇒ Words

Sentence → Colloc$^+$
Colloc → Word$^+$
Word → 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; ≈ Goldwater's bigram model)

colloc-syll: Collocations ⇒ Words ⇒ Syllables

Sentence → Colloc^+

Colloc → Word^+

Word → SyllableIF

Syllable → (Onset) Rhyme

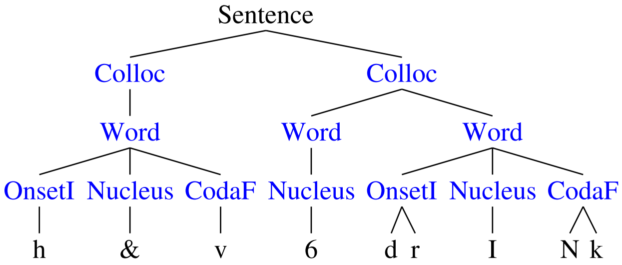
Word → SyllableI (Syllable) (Syllable) SyllableF

Onset → Consonant^+

Rhyme → Nucleus (Coda)

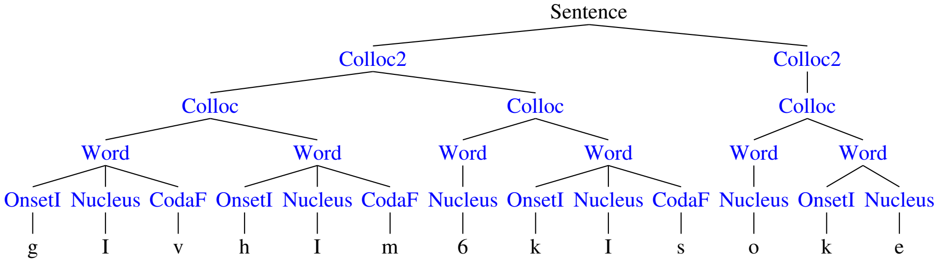
Nucleus → Vowel^+

Coda → Consonant^+



- With 2 Collocation levels, f-score = 87%

colloc-syll: Collocations ⇒ Words ⇒ Syllables



Bayesian inference for PYP parameters

- Adaptor grammars have 1 (CRP) or 2 (PYP) hyper-parameters for each adapted non-terminal X
- Previous work used CRP adaptors with *tied parameters*
- Bayesian prior: for each adapted nonterminal X

$$\begin{aligned} a_X &\sim \mathit{Beta}(1,1) \\ b_X &\sim \mathit{Gamma}(10,0.1) \end{aligned}$$

  - $\mathit{Gamma}(10, 0.1)$ is a vague Gamma prior (MacKay 2003)
  - permits a_X and b_X to *vary with adapted nonterminal X*
- Estimate with *slice sampling* (no proposal distribution; Neal 2003)
- *Biggest improvement on complex models*, e.g., colloc-syll:
  - tied parameters 78% token f-score
  - $a_X = 0$, sampling b_X (i.e., CRP adaptors) 84% token f-score
  - sampling a_X and b_X (i.e., PYP adaptors) 87% token f-score

Outline

Inference for PCFG rule probabilities

Learning grammar rules (not just probabilities)

Adaptor grammars

Non-parametric inference in word segmentation
- Priors on Adaptor Grammar PYP parameters

Unsupervised inference of Adaptor Grammars
- Modal word segmentation
- Random vs incremental initialization
- Table label resampling

Conclusion

Unsupervised inference via Gibbs sampling

- Observations (terminal strings) $\boldsymbol{x} = (x_1, \ldots, x_n)$
  Hidden labels (parse trees) $\boldsymbol{t} = (t_1, \ldots, t_n)$
  Probabilistic model (adaptor grammar) $\mathrm{P}(\boldsymbol{x}, \boldsymbol{t})$
- Gibbs sampling algorithm:

  initialize $\boldsymbol{t}$ somehow (e.g., random trees)
  repeat forever:
      pick an index $j \in 1, \ldots, n$ at random
      replace t_j with a random sample from $\mathrm{P}(t \mid x_j, \boldsymbol{t}_{-j})$
          where $\boldsymbol{t}_{-j} = (t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_n)$
- After *burn-in* the samples $\boldsymbol{t}$ are distributed according to $\mathrm{P}(\boldsymbol{t} \mid \boldsymbol{x})$

Finding the modal word segmentation

- Previous work decoded using last sampled trees (2,000 epochs)
- After burn-in, samples are distributed according to $\mathrm{P}(\boldsymbol{t} \mid \boldsymbol{x})$
  $\Rightarrow$ use samples to identify *modal word segmentation*
- Modal decoding:
  - For each sentence x_i save *sample parses* $\boldsymbol{s}_i = (s_i^{(1)}, \ldots, s_i^{(800)})$
    (every 10th epoch from epochs 1,000–2,000 from 8 runs)
  - Compute *word segmentations* $\boldsymbol{w}_i = (w_i^{(1)}, \ldots, w_i^{(800)})$ from parses
  - Compute *modal segmentation* $\widehat{w}_i = \mathit{argmax}_w\, n_w(\boldsymbol{w}_i)$,
    where $n_w(\boldsymbol{w}_i)$ is the number of times w appears in $\boldsymbol{w}_i$
- Improves word segmentation token f-score in all models

Model	Average	Max-modal
unigram	55%	56%
colloc	74%	76%
colloc-syll	85%	87%

- Goodman (1998) max-marginal decoding should also be possible

Random vs incremental initialization

- The Gibbs sampler parse trees $\boldsymbol{t}$ needs to be initialized somehow

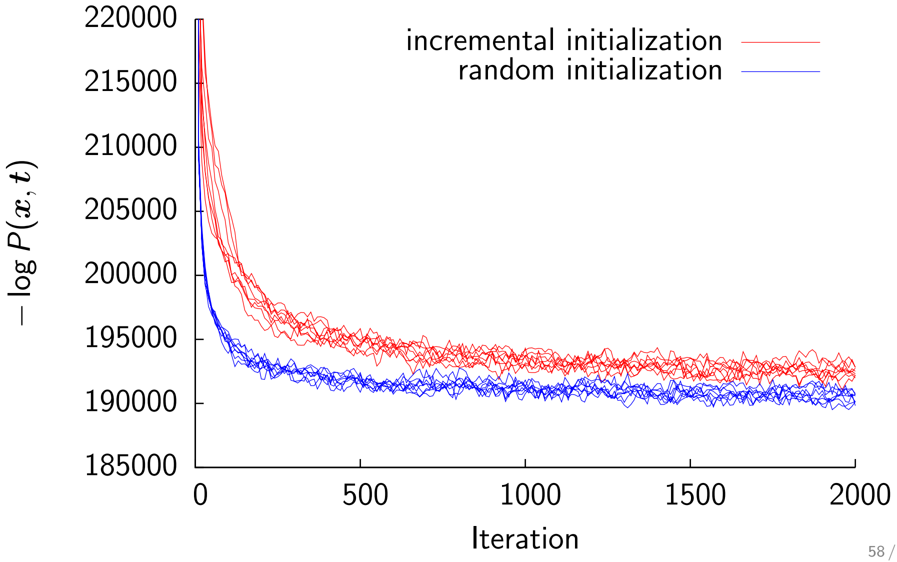
  Random initialization: Assign each string x_i a random parse t_i generated by base PCFG

  Incremental initialization: Sample t_i from $\mathrm{P}(t \mid x_i, \boldsymbol{t}_{1:i-1})$
- Incremental initialization is easy to implement in a Gibbs sampler
- Incremental initialization improves token f-score in all models, especially on simple models

Model	Random	Incremental
unigram	56%	81%
colloc	76%	86%
colloc-syll	87%	89%

but see caveats on next slide!

Incremental initialization produces low-probability parses



Why incremental initialization produces low-probability parses

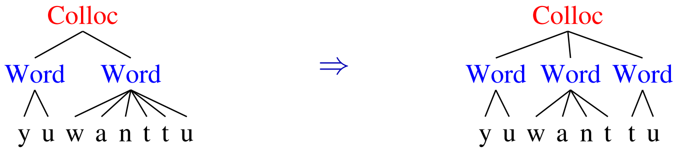
- Incremental initialization produces sample parses $\boldsymbol{t}$ with lower probability $\mathrm{P}(\boldsymbol{t} \mid \boldsymbol{x})$
- Possible explanation: (Goldwater's 2006 analysis of Brent's model)
  - All the models tend to *undersegment* (i.e., find collocations instead of words)
  - Incremental initialization *greedily searches for common substrings*
  - Shorter strings are more likely to be recurr early than longer ones

Table label resampling

- Each adapted non-terminal has a CRP with tables labelled with parses
- "Rich get richer" ⇒ resampling a sentence's parse reuses the same cached subtrees
- *Resample table labels* as well sentence parses
  - A table label may be used in many sentence parses
  - ⇒ Resampling a single table label may change the parses of a single sentence
  - ⇒ table label resampling can improve mobility with grammars with a hierarchy of adapted non-terminals
- Essential for grammars with a complex hierarchical structure

Table label resampling example

Label on table in Chinese Restaurant for colloc



Resulting changes in parse trees

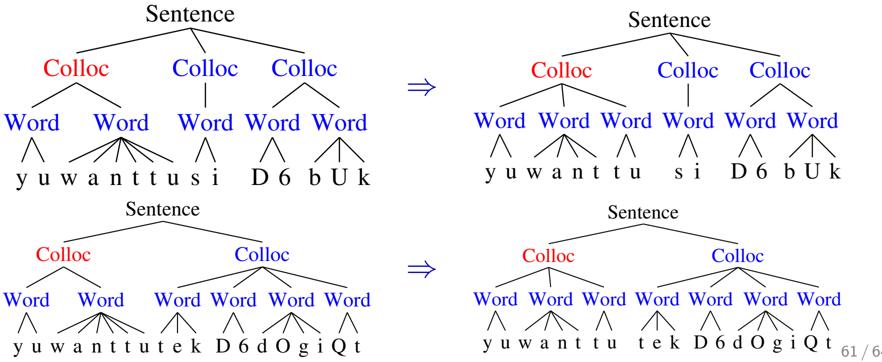
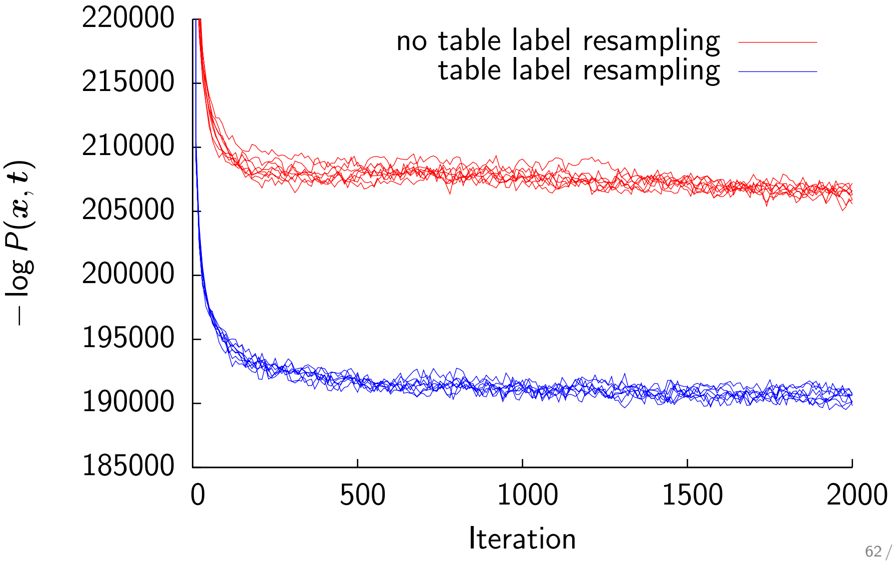


Table label resampling produces much higher-probability parses



Outline

Inference for PCFG rule probabilities

Learning grammar rules (not just probabilities)

Adaptor grammars

Non-parametric inference in word segmentation
- Priors on Adaptor Grammar PYP parameters

Unsupervised inference of Adaptor Grammars
- Modal word segmentation
- Random vs incremental initialization
- Table label resampling

Conclusion

Conclusion

- MCMC sampling is a natural approach to inference in non-parametric models
- Put Bayesian priors on PYP hyperparameters and slice sample
  - improves colloc-syll model by 9%
- Modal decoding from multiple samples
  - improves colloc-syll model by 2%
- Random initialization instead of incremental initialization
  - hurts colloc-syll model by 2%, but produces higher probability parses
- Table label resampling in hierarchical CRPs/PYPs
  - improves colloc-syll model by 20%
- Taken together, performance of colloc-syll model improves from last year's 78% to 87% token f-score.
- Software available from cog.brown.edu/~mj