# Parsing and Learning

International Workshop on Parsing Technology

Mark Johnson

Brown University / Macquarie University

October 2009

# Parsing and Learning

- Parsing and grammar induction traditionally viewed as distinct
  - Unknown words and phrases $\Rightarrow$ parser must be prepared to adapt
- Most grammar learning systems are *error-driven*
  - parsing is a central component of most learning algorithms
- Traditional statistical learning algorithms learn *rule probabilities*
- Learning grammar rules
  - via “generate and test”
  - via *non-parametric Bayes* $\Rightarrow$ *adaptor grammars*

# Talk outline

- Learning PCFG rule probabilities from terminal strings
  - collapsed Gibbs samplers explictly sample parses, but don't explicitly represent rule probabilities
- Non-parametric extensions to PCFGs with unboundedly many possible rules
  - adaptor grammars learn probabilities of arbitrary subtrees
  - collapsed Gibbs sampler only needs to represent sample parses, and not probabilities of infinitely many rules
- Adaptor grammars can be used to learn:
  - simple concatenative morphology
  - unsupervised word segmentation
  - collocations in topic models
  - structure in names

# Outline

Learning rule probabilities

Learning grammar rules (not just probabilities)

Chinese Restaurant Processes

Adaptor grammars

Bayesian inference for adaptor grammars

Adaptor grammars for unsupervised word segmentation

Topic Collocation models using adaptor grammars

Adaptor grammars for named entities

Conclusion

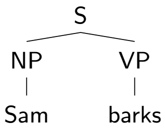
Extending Adaptor Grammars

# Probabilistic context-free grammars

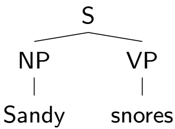
- Rules in *Context-Free Grammars* (CFGs) expand nonterminals into sequences of terminals and nonterminals
- A *Probabilistic CFG* (PCFG) associates each nonterminal with a multinomial distribution over the rules that expand it
- Probability of a tree is the *product of the probabilities of the rules*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

| Rule $r$ | $\theta_r$ |
|---|---|
| S $\rightarrow$ NP VP | 1.0 |
| NP $\rightarrow$ Sam | 0.75 |
| VP $\rightarrow$ barks | 0.6 |

| Rule $r$ | $\theta_r$ |
|---|---|
| | |
| NP $\rightarrow$ Sandy | 0.25 |
| VP $\rightarrow$ snores | 0.4 |

P ( 

) = 0.45

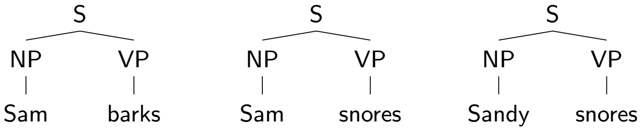
P ( 

) = 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 $r$ | $n_r$ | $\theta_r$ |
|---|---|---|
| S $\rightarrow$ NP VP | 3 | 1.0 |
| NP $\rightarrow$ Sam | 2 | 0.66 |
| VP $\rightarrow$ barks | 1 | 0.33 |

| Rule $r$ | $n_r$ | $\theta_r$ |
|---|---|---|
| | | |
| NP $\rightarrow$ Sandy | 1 | 0.33 |
| VP $\rightarrow$ snores | 2 | 0.66 |

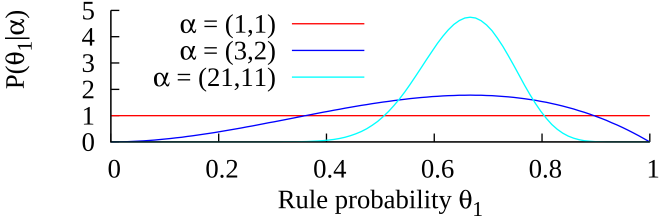
- But MLE is often *overly certain*, especially with sparse data
  - E.g., "accidental zeros" $n_r = 0 \Rightarrow \theta_r = 0$.

# Bayesian estimation from visible parses

- Bayesian estimators estimate a *distribution* over rule probabilities

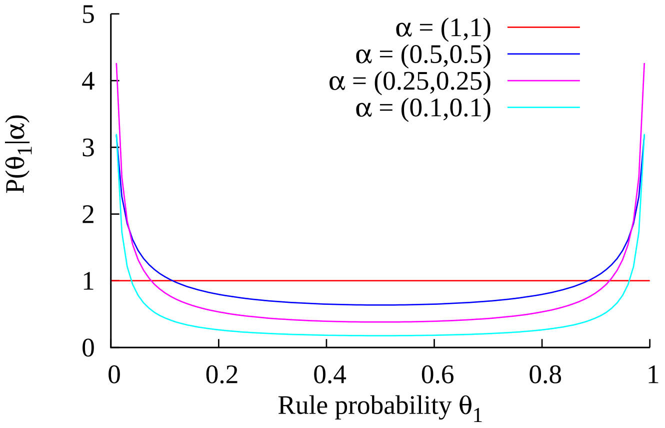
$$\underbrace{\mathsf{P}(\boldsymbol{\theta} \mid \boldsymbol{n})}_{\text{Posterior}} \propto \underbrace{\mathsf{P}(\boldsymbol{n} \mid \boldsymbol{\theta})}_{\text{Likelihood}} \underbrace{\mathsf{P}(\boldsymbol{\theta})}_{\text{Prior}}$$

- *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 $x_i$.

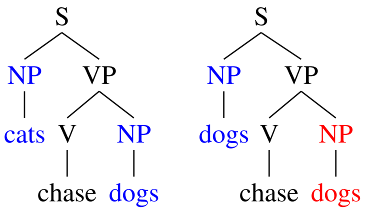
# Collapsed Gibbs samplers

- *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

Learning rule probabilities

Learning grammar rules (not just probabilities)

Chinese Restaurant Processes

Adaptor grammars

Bayesian inference for adaptor grammars

Adaptor grammars for unsupervised word segmentation

Topic Collocation models using adaptor grammars

Adaptor grammars for named entities

Conclusion

Extending Adaptor Grammars

# 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<br>
  Repeat:<br>
      re-estimate rule probabilities from strings<br>
      prune low probability rules<br>
      *propose additional potentially useful rules*
- Non-parametric Bayesian methods seem to provide a more systematic approach

# Non-parameteric Bayesian extensions to PCFGs

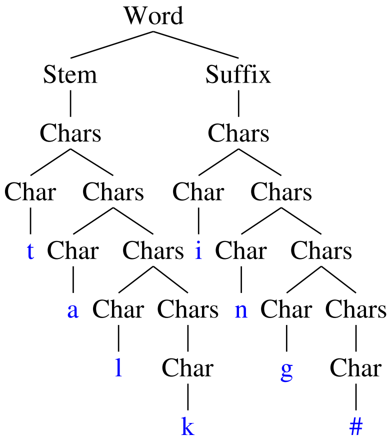
- Non-parametric $\Rightarrow$ no fixed set of parameters
- Two obvious non-parametric extensions to PCFGs:
  - let the set of non-terminals grow unboundedly
    - given an initial grammar with coarse-grained categories, split non-terminals into more refined categories
      $S_{12} \rightarrow NP_7\, VP_4$ instead of $S \rightarrow NP\, VP$.
    - 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

| | | | | | |
|---|---|---|---|---|---|
| Word | → | Stem Suffix | Chars | → | Char |
| Stem | → | Chars | Chars | → | Char Chars |
| Suffix | → | Chars | Char | → | a \| b \| c \| . . . |



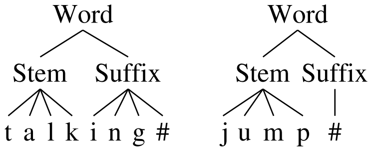

- Grammar's trees can represent any segmentation of words into stems and suffixes

⇒ 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

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



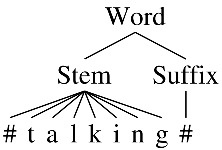

- A rule for each morpheme
  $\Rightarrow$ "PCFG" can represent probability of each morpheme
- *Unbounded number of possible rules, so this is not a PCFG*
  - not a practical problem, as only a finite set of rules could possibly be used in any particular data set

# 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



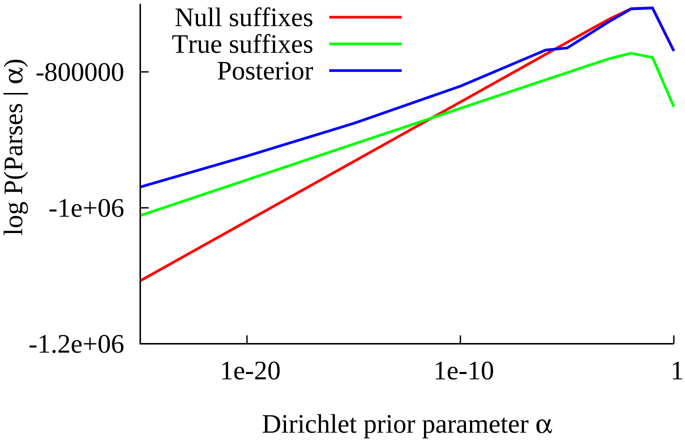
# Forcing generalization using Dirichlet priors

- 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 $\alpha \rightarrow 0$
  - 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 | |

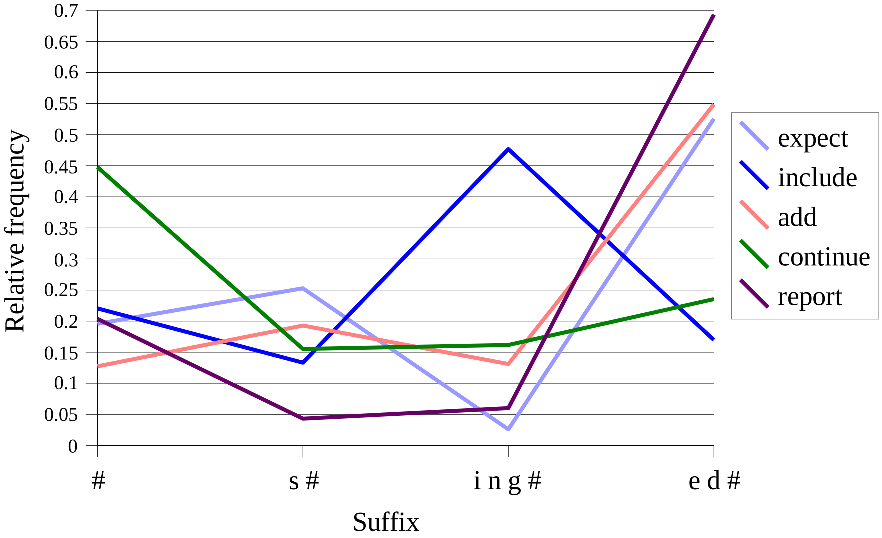
# Log posterior for models on token data




- Correct solution is nowhere near as likely as posterior

⇒ 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

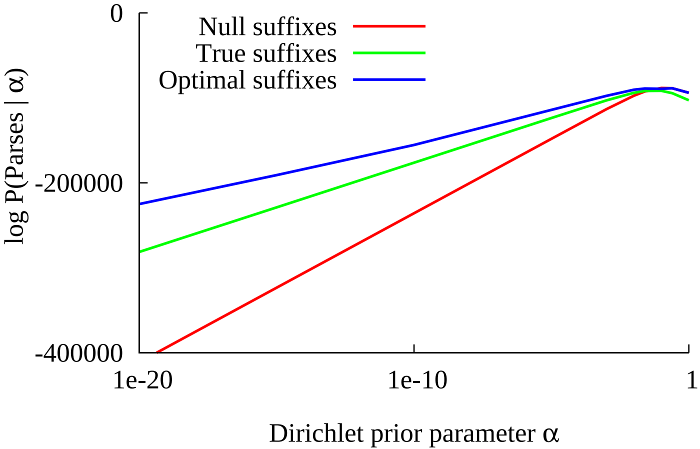
$$
\begin{aligned}
\text{Data} &= \text{“the cat chased the other cat”} \\
\text{Tokens} &= \text{“the”}, \text{“cat”}, \text{“chased”}, \text{“the”}, \text{“other”}, \text{“cat”} \\
\text{Types} &= \text{“the”}, \text{“cat”}, \text{“chased”}, \text{“other”}
\end{aligned}
$$

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

# From Dirichlet-multinomials to Chinese Restaurant Processes

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

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

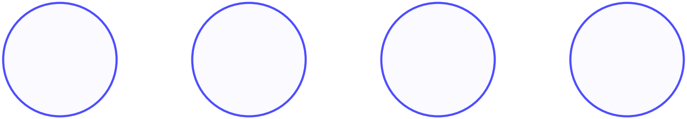
- Let $m \to \infty$

$$\begin{aligned}
\mathrm{P}(Z_{n+1} = k \mid \boldsymbol{z}) &\;\propto\; n_k(\boldsymbol{z}) \text{ if } k \text{ appears in } \boldsymbol{z} \\
\mathrm{P}(Z_{n+1} \not\in \boldsymbol{z} \mid \boldsymbol{z}) &\;\propto\; \alpha
\end{aligned}$$

- If outcomes are exchangable $\Rightarrow$ number in order of occurence $\Rightarrow$ *Chinese Restaurant Process*

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

# Chinese Restaurant Process (0)

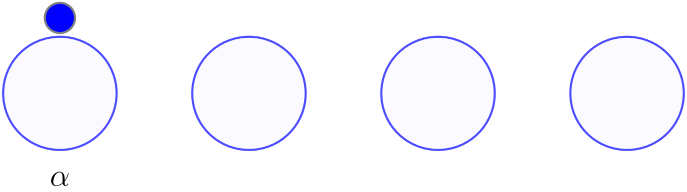


- Customer $\rightarrow$ table mapping $\boldsymbol{z} =$
- $\mathrm{P}(\boldsymbol{z}) = 1$

- Next customer chooses a table according to:

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

# Chinese Restaurant Process (1)

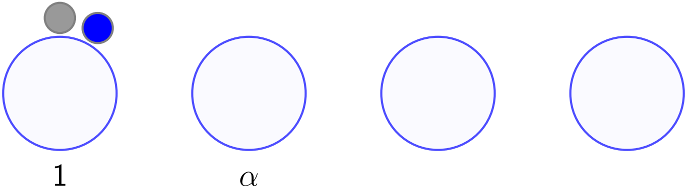



- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha$

- Next customer chooses a table according to:

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

# Chinese Restaurant Process (2)

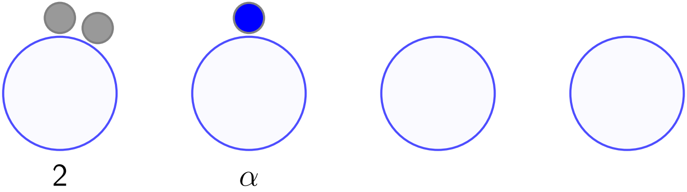



- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha)$

- Next customer chooses a table according to:

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

# Chinese Restaurant Process (3)

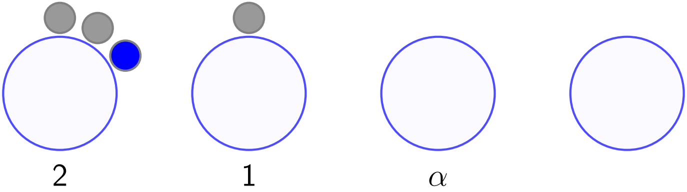



- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1, 2$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha) \times \alpha/(2+\alpha)$

- Next customer chooses a table according to:

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

# Chinese Restaurant Process (4)

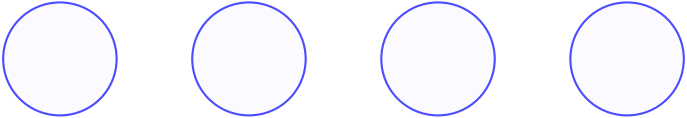



- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1, 2, 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha) \times \alpha/(2+\alpha) \times 2/(3+\alpha)$

- Next customer chooses a table according to:

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

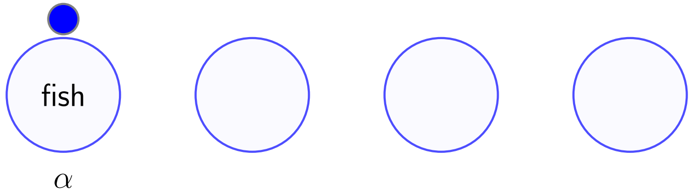
# Labeled Chinese Restaurant Process (0)



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

- *Base distribution* $\mathrm{P}_0(Y)$ 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 $x_i = y_{z_i}$

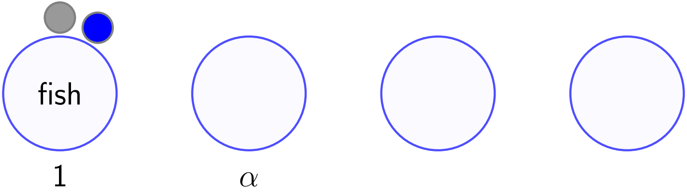
# Labeled Chinese Restaurant Process (1)




- Table $\rightarrow$ label mapping $\boldsymbol{y}$ = fish
- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1$
- Output sequence $\boldsymbol{x}$ = fish
- $\mathrm{P}(\boldsymbol{x}) = \alpha/\alpha \times \mathrm{P}_0(\text{fish})$

- *Base distribution* $\mathrm{P}_0(Y)$ 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 $x_i = y_{z_i}$

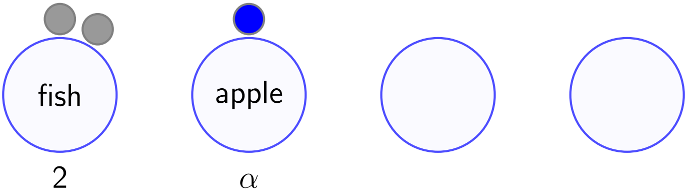
# Labeled Chinese Restaurant Process (2)




- Table → label mapping $\boldsymbol{y}$ = fish
- Customer → table mapping $\boldsymbol{z} = 1, 1$
- Output sequence $\boldsymbol{x}$ = fish,fish
- $\mathrm{P}(\boldsymbol{x}) = \mathrm{P}_0(\text{fish}) \times 1/(1+\alpha)$

- *Base distribution* $\mathrm{P}_0(Y)$ 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 $x_i = y_{z_i}$

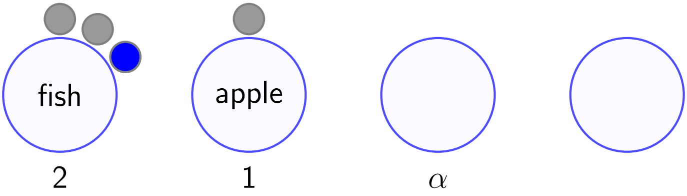
# Labeled Chinese Restaurant Process (3)




- Table $\rightarrow$ label mapping $\boldsymbol{y}$ = fish,apple
- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1, 2$
- Output sequence $\boldsymbol{x}$ = fish,fish,apple
- $\mathrm{P}(\boldsymbol{x}) = \mathrm{P}_0(\text{fish}) \times 1/(1+\alpha) \times \alpha/(2+\alpha)\mathrm{P}_0(\text{apple})$

- *Base distribution* $\mathrm{P}_0(Y)$ 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 $\boldsymbol{y}$ = fish,apple
- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1, 2$
- Output sequence $\boldsymbol{x}$ = fish,fish,apple,fish
- $\mathrm{P}(\boldsymbol{x}) = \mathrm{P}_0(\text{fish}) \times 1/(1+\alpha) \times \alpha/(2+\alpha)\mathrm{P}_0(\text{apple}) \times 2/(3+\alpha)$

- *Base distribution* $\mathrm{P}_0(Y)$ 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* $\alpha$ controls how likely a new outcome is
  - CRPs exhibit a *rich get richer* power-law behaviour
- *Pitman-Yor Processes* (PYPs) generalize CRPs by adding an additional parameter (each PYP has $a$ and $b$ parameters)
  - PYPs can describe a wider range of distributions than CRPs
- *Labeled CRPs and PYPs* use a *base distribution* to label each table
  - base distribution can have *infinite support*
  - concentrates mass on a countable subset

# Labeled Chinese restaurants and Dirichlet processes

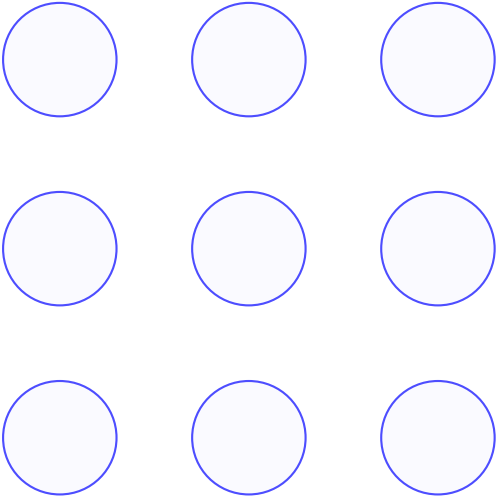
- A labeled Chinese restaurant processes maps a *base distribution* $\mathsf{P}_B$ to a stream of samples from a different distribution with the same support
- CRPs specify the *conditional distribution* of the next outcome given the previous ones
- Each CRP run can produce a different distribution over labels
- It defines a mapping from $\alpha$ and $\mathsf{P}_B$ to a *distribution over distributions* $\mathrm{DP}(\alpha, \mathsf{P}_B)$
- $\mathrm{DP}(\alpha, \mathsf{P}_B)$ is called a *Dirichlet process* (DP) with *concentration parameter* $\alpha$ and *base distribution* $\mathsf{P}_B$
- The base distribution $\mathsf{P}_B$ can itself be defined by a DP
  $\Rightarrow$ *hierarchy* of DPs

# Outline

# 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* $\alpha_A$
- 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 for stem-suffix morphology (0)



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

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

<u>Suffix</u> $\rightarrow$ Phoneme$^{\star}$

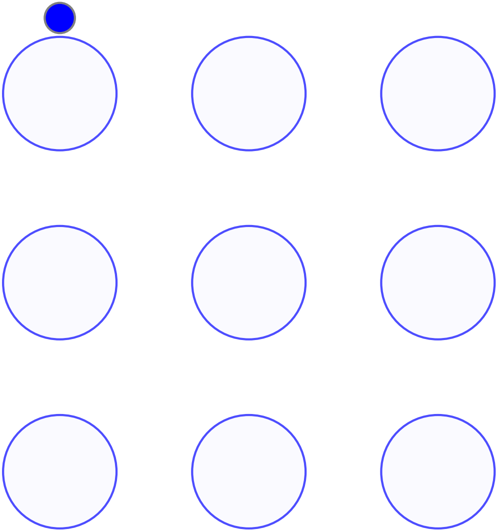
Generated words:

# Adaptor grammar for stem-suffix morphology (1a)

Word → Stem Suffix

Stem → Phoneme$^{+}$

Suffix → Phoneme$^{\star}$



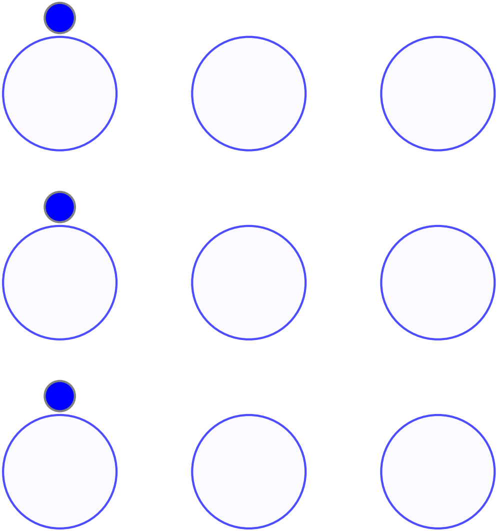
Generated words:

# Adaptor grammar for stem-suffix morphology (1b)

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

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

<u>Suffix</u> $\rightarrow$ Phoneme$^\star$



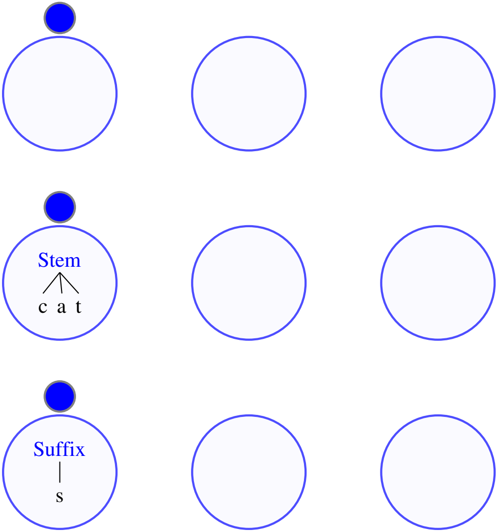
Generated words:

# Adaptor grammar for stem-suffix morphology (1c)

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

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

<u>Suffix</u> $\rightarrow$ Phoneme$^{\star}$



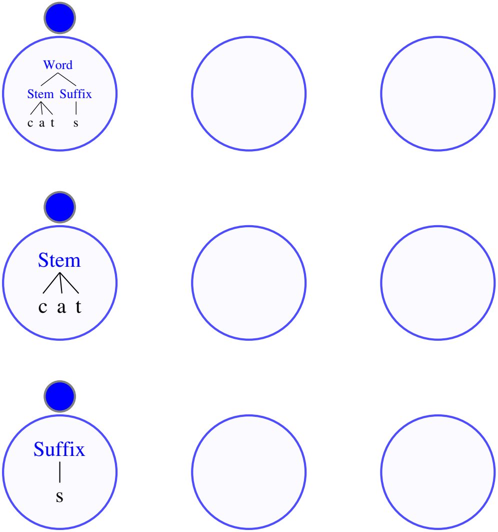

Generated words:

# Adaptor grammar for stem-suffix morphology (1d)

Word $\longrightarrow$ Stem Suffix

Stem $\longrightarrow$ Phoneme$^{+}$

Suffix $\longrightarrow$ Phoneme$^{\star}$



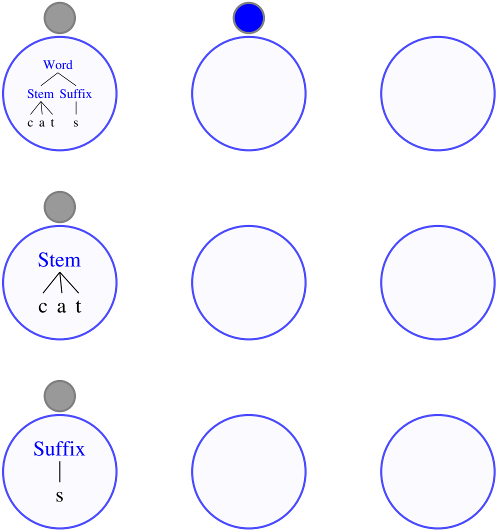

Generated words: cats

# Adaptor grammar for stem-suffix morphology (2a)

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

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

<u>Suffix</u> $\rightarrow$ Phoneme$^\star$



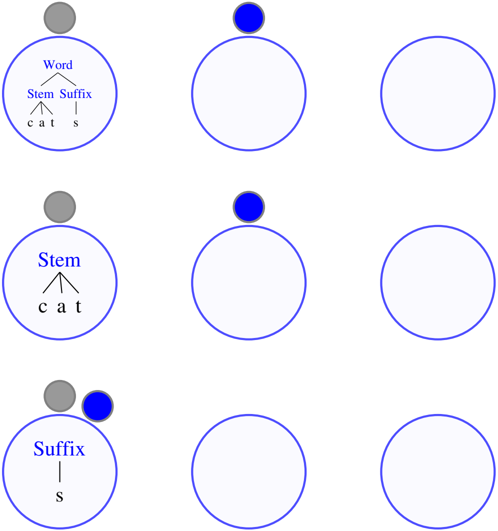

Generated words: cats

# Adaptor grammar for stem-suffix morphology (2b)

Word → Stem Suffix

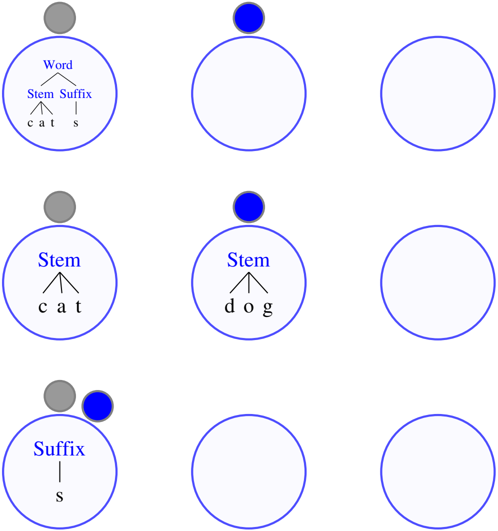
Stem → Phoneme$^+$

Suffix → Phoneme$^\star$




Generated words: cats

# Adaptor grammar for stem-suffix morphology (2c)




Word $\longrightarrow$ Stem Suffix

Stem $\longrightarrow$ Phoneme$^+$

Suffix $\longrightarrow$ Phoneme$^\star$

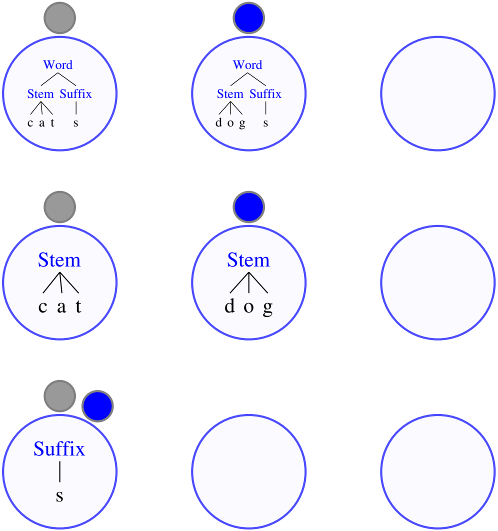
Generated words: cats

# Adaptor grammar for stem-suffix morphology (2d)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme$^+$

Suffix $\rightarrow$ Phoneme$^\star$



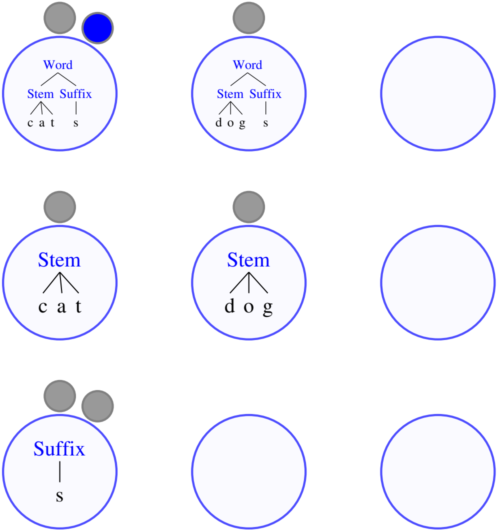

Generated words: cats, dogs

# Adaptor grammar for stem-suffix morphology (3)

<u>Word</u> → Stem Suffix

<u>Stem</u> → Phoneme$^{+}$

<u>Suffix</u> → Phoneme$^{\star}$




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

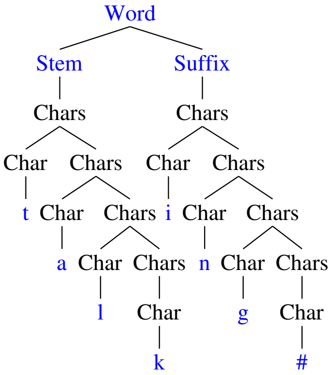
- Possible trees generated by CFG rules
  but the probability of each adapted tree is estimated separately
- Probability of adapted nonterminal $A$ expanding to subtree $\tau$ is proportional to:
  - the number of times $\tau$ was seen before
    $\Rightarrow$ "rich get richer" dynamics (Zipf distributions)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

# 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
  $\Rightarrow$ higher in Bayesian hierarchy



# Outline

# Estimating adaptor grammars

- Need to estimate:
  - parse trees $\boldsymbol{t} = (t_1, \ldots, t_n)$ for strings $\boldsymbol{x} = (x_1, \ldots, x_n)$
  - cached subtrees $\tau$ for adapted nonterminals
  - DP parameters $\boldsymbol{\alpha}$ for adapted nonterminals
  - probabilities $\boldsymbol{\theta}$ of base grammar rules
- Collapsed Metropolis-within-Gibbs sampler for parse trees
  - sample parse tree $t_i$ from $\mathrm{P}(t \mid x_i, \boldsymbol{t}_{-i})$
    where $\boldsymbol{t}_{-i} = (t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$
  - sampling directly from conditional distribution of parses seems intractable (just as for PCFGs)
  - construct PCFG *proposal grammar* $G'(\boldsymbol{t}_{-i})$ on the fly
    $G'(\boldsymbol{t}_{-i})$ contains a rule for each cached subtree $\tau$
    $\Rightarrow$ grammar rules change while sampling
  - MH accept/reject determines whether to update $t_i$

# Collapsed Metropolis-within-Gibbs sampling

- Input: terminal strings $\boldsymbol{x} = (x_1, \ldots, x_n)$
- Output: stream of sample parse trees $\boldsymbol{t} = (t_1, \ldots, t_n)$
- Metropolis-within-Gibbs sampling algorithm:

  initialize $\boldsymbol{t}$ somehow (e.g., random trees)
  repeat forever:

    pick a sentence $x_i$ at random
    construct PCFG proposal grammar $G'(\boldsymbol{t}_{-i})$ on the fly
    sample parse tree $t$ from $\mathrm{P}(t \mid x_j, G'(\boldsymbol{t}_{-i}))$
    update $t_i$ with $t$ according to MH accept-reject procedure

- After *burn-in* the samples $\boldsymbol{t}$ are distributed according to $\mathrm{P}(\boldsymbol{t} \mid \boldsymbol{x})$

# Outline

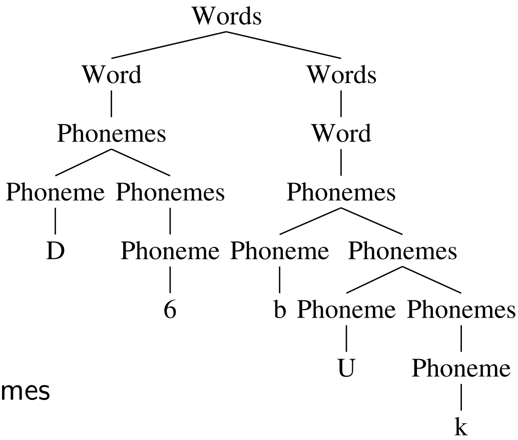
# Unsupervised word segmentation

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

y △ u ▲ w △ a △ n △ t ▲ t △ u ▲ s △ i ▲ D △ 6 ▲ b △ U △ k

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

# Word segmentation with PCFGs (1)




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

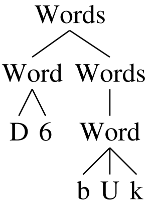
which abbreviates

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

- PCFG trees are adequate for representing word segmentation, but PCFG rules are *too small a unit of generalization* to learn words

# Word segmentation with PCFGs (1)

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

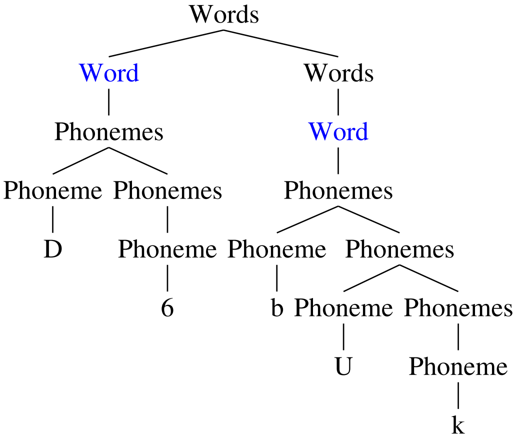



- But now there are an infinite number of PCFG rules!
  - once we see our (finite) training data, only finitely many are useful
  - $\Rightarrow$ the set of parameters (rules) should be chosen based on training data

# Unigram word segmentation adaptor grammar

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

- *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

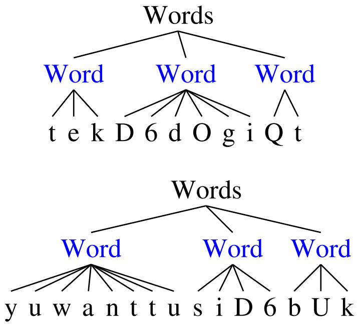
$$\text{Words} \to \text{Word}^+$$
$$\underline{\text{Word}} \to \text{Phon}^+$$

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

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

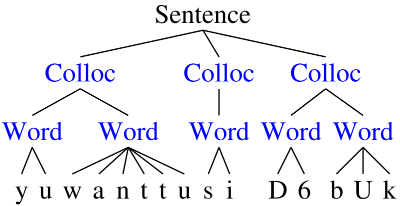
# unigram: Words

- 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 $\Rightarrow$ Words

$$\begin{aligned}&\text{Sentence} \to \text{Colloc}^+\\&\underline{\text{Colloc}} \to \text{Word}^+\\&\underline{\text{Word}} \to \text{Phon}^+\end{aligned}$$




- 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)

# colloc-syll: Collocations $\Rightarrow$ Words $\Rightarrow$ Syllables

Sentence $\rightarrow$ Colloc$^+$

Colloc $\rightarrow$ Word$^+$

Word $\rightarrow$ SyllableIF

Syllable $\rightarrow$ (Onset) Rhyme

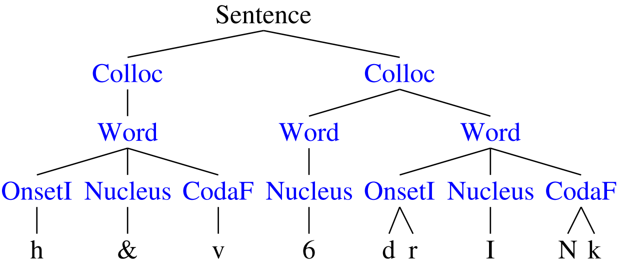
Word $\rightarrow$ SyllableI (Syllable) (Syllable) SyllableF

Onset $\rightarrow$ Consonant$^+$

Rhyme $\rightarrow$ Nucleus (Coda)

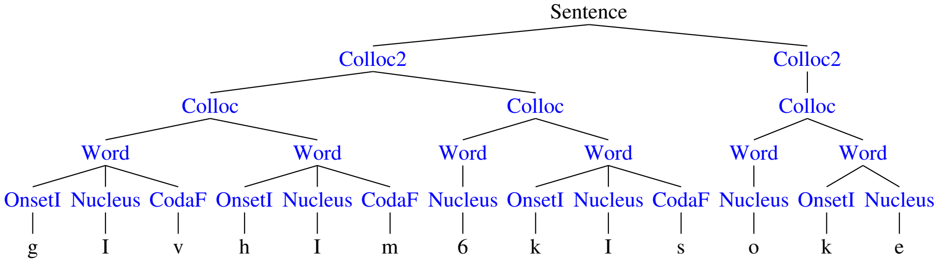
Nucleus $\rightarrow$ Vowel$^+$

Coda $\rightarrow$ Consonant$^+$




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

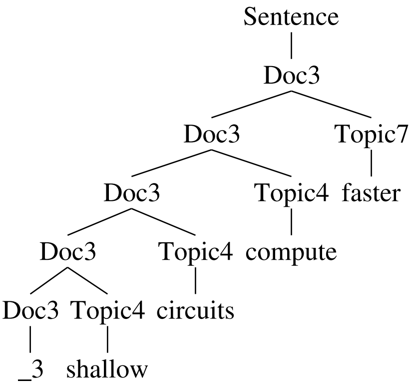
# colloc-syll: Collocations ⇒ Words ⇒ Syllables



# Outline

# LDA topic models as PCFGs

- Each document $i$ generates a distribution over $m$ topics
- Each topic $j$ generates a (unigram) distribution over vocabulary $\mathcal{X}$.
- Preprocess input by prepending a document id to every sentence




$$
\begin{array}{lll}
\text{Sentence} \rightarrow & \text{Doc}_i & i \in 1, \ldots, n \\
\text{Doc}_i \rightarrow & \_i & i \in 1, \ldots, n \\
\text{Doc}_i \rightarrow & \text{Doc}_i \ \text{Topic}_j & i \in 1, \ldots, n; j \in 1, \ldots, m \\
\text{Topic}_j \rightarrow & x & j \in 1, \ldots, m; x \in \mathcal{X}
\end{array}
$$

# Estimated PCFG for LDA topic model

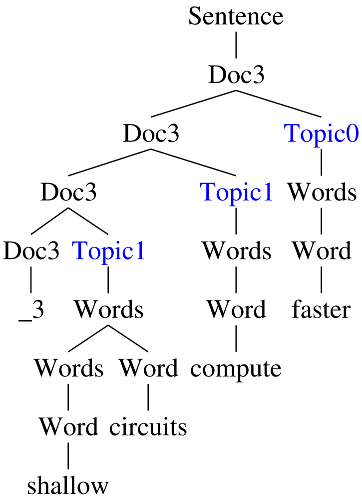
| Rule | Count |
|---|---|
| $\mathrm{Doc}_3 \rightarrow \mathrm{Doc}_3\ \mathrm{Topic}_4$ | 737 |
| $\mathrm{Doc}_3 \rightarrow \mathrm{Doc}_3\ \mathrm{Topic}_3$ | 392 |
| $\mathrm{Doc}_3 \rightarrow \mathrm{Doc}_3\ \mathrm{Topic}_9$ | 263 |
| $\mathrm{Doc}_3 \rightarrow \mathrm{Doc}_3\ \mathrm{Topic}_7$ | 254 |
| ... | |
| $\mathrm{Topic}_4 \rightarrow$ function | 3576 |
| $\mathrm{Topic}_4 \rightarrow$ functions | 2195 |
| $\mathrm{Topic}_4 \rightarrow$ error | 1614 |
| $\mathrm{Topic}_4 \rightarrow$ algorithm | 1588 |
| ... | |
| $\mathrm{Topic}_7 \rightarrow$ network | 13018 |
| $\mathrm{Topic}_7 \rightarrow$ input | 6716 |
| $\mathrm{Topic}_7 \rightarrow$ training | 6078 |
| $\mathrm{Topic}_7 \rightarrow$ learning | 5176 |
| ... | |

# Finding topic collocations with adaptor grammars

- Let each $\text{Topic}_j$ expand to a *sequence* of words
- Adapt each $\text{Topic}_j$

⇒ learn topic's likely collocations

$$\begin{aligned}
\text{Sentence} &\rightarrow \text{Doc}_i \\
\text{Doc}_i &\rightarrow _i \\
\text{Doc}_i &\rightarrow \text{Doc}_i\ \text{Topic}_j \\
\underline{\text{Topic}_j} &\rightarrow \text{Words} \\
\text{Words} &\rightarrow \text{Word} \\
\text{Words} &\rightarrow \text{Words}\ \text{Word}
\end{aligned}$$




- Would be easy to make a hierarchical adaptor grammar model that shares collocations between topics

# Outline

# 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 $\rightarrow$ (A0) (A1) ... (A6) | NP $\rightarrow$ (B0) (B1) ... (B6) |
| $\underline{\text{A0}} \rightarrow \text{Word}^+$ | $\underline{\text{B0}} \rightarrow \text{Word}^+$ |
| ... | ... |
| $\underline{\text{A6}} \rightarrow \text{Word}^+$ | $\underline{\text{B6}} \rightarrow \text{Word}^+$ |
| NP $\rightarrow \text{Unordered}^+$ | $\underline{\text{Unordered}} \rightarrow \text{Word}^+$ |

(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)

- See Elsner et al (2009) for details

# Outline

# Summary and future work

- Parsing and learning are intimately related
  - most methods for grammar learning involve repeated parsing
- *Non-parametric generalizations* of PCFGs introduce an infinite number of potential rules
  - *Adaptor Grammars* learn entire subtrees
  - can define a range of interesting models
- Collapsed Gibbs samplers are a natural method for non-parametric inference
  - a finite number of samples can only involve a finite number of structures
  - efficient sampling using Metropolis-within-Gibbs with PCFG proposals
- *Non-parametric Bayesian approaches provide a principled approach to learning grammar rules as well as their probabilities*

# Outline

# 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* $A \to \beta$, where $A \in N$ and $\beta \in (N \cup W)^\star$
- 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:

$$\mathcal{T}_A = \bigcup_{A \to B_1 \ldots B_n \in R_A} \textsc{Tree}_A(\mathcal{T}_{B_1}, \ldots, \mathcal{T}_{B_n})$$

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

$$\textsc{Tree}_A(\mathcal{T}_{B_1}, \ldots, \mathcal{T}_{B_n}) = \left\{ \begin{array}{c} A \\ \widehat{t_1 \ \ldots \ t_n} \end{array} : \begin{array}{l} t_i \in \mathcal{T}_{B_i}, \\ i = 1, \ldots, n \end{array} \right\}$$

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 $\boldsymbol{\theta}$, where:

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

It defines distributions $G_A$ over trees $\mathcal{T}_A$ for $A \in N \cup W$:

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

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

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

$\mathrm{TD}_A(G_1,\ldots,G_n)$ 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 $A \in N$, $\alpha_A$ is the parameter of the Dirichlet process associated with $A$.

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

$$H_A = \sum_{A \to B_1 \ldots B_n \in R_A} \theta_{A \to B_1 \ldots B_n} \mathrm{TD}_A(G_{B_1}, \ldots, G_{B_n})$$

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

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

$$H_A = \sum_{A \to B_1 \ldots B_n \in R_A} \theta_{A \to B_1 \ldots B_n} \mathrm{TD}_A(G_{B_1}, \ldots, G_{B_n})$$

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

# Adaptive fragment grammars

- Disentangle syntactic and Bayesian hierarchy
  - *Adaptive metagrammar* generates *fragment distributions*
  - which plug together as in tree substitution grammar
- Tree fragment sets $\mathcal{P}_A, A \in N$ are smallest sets satisfying:

$$\mathcal{P}_A \;=\; \bigcup_{A \to B_1 \ldots B_n \in R_A} \textsc{Tree}_A(\{B_1\} \cup \mathcal{P}_{B_1}, \ldots, \{B_n\} \cup \mathcal{P}_{B_n})$$

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

$$G_A \;=\; \sum_{\underset{B_1 \ldots B_n}{\overset{A}{\triangle}} \in \mathcal{P}_A} F_A\Big(\underset{B_1 \ldots B_n}{\overset{A}{\triangle}}\Big) \;\; \mathrm{TD}_{\underset{B_1 \ldots B_n}{\overset{A}{\triangle}}}(G_{B_1}, \ldots, G_{B_n})$$

- A fragment grammar generates the distribution $G_S$

# Adaptive fragment distributions

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

$$H_A \;=\; \sum_{A \to B_1 \ldots B_n \in R_A} \theta_{A \to B_1 \ldots B_n} \mathrm{TD}_A(\eta\, \delta_{B_1} + (1-\eta) H_{B_1}, \ldots)$$

  where $\eta$ is the *fragment exit probability*
- Obtain $F_A$ by *adapting the $H_A$ distribution*

$$F_A \;\sim\; \mathrm{DP}(\alpha_A, H_A)$$

- This construction can be *iterated*, i.e., replace $\boldsymbol{\theta}$ with another fragment distribution
- Question: if we iterate this, when does the *fixed point* exist, and what is it?