Learning rules with Adaptor Grammars

Mark Johnson

joint work with Sharon Goldwater and Tom Griffiths

November 2009

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.*"

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

Language acquisition as Bayesian inference

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

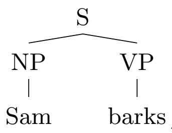
- Likelihood measures how well grammar describes data
- Prior expresses knowledge of grammar before data is seen
  - can be very specific (e.g., Universal Grammar)
  - can be very general (e.g., prefer shorter grammars)
- Posterior is a *distribution* over grammars
  - captures *learner's uncertainty* about which grammar is correct

Outline

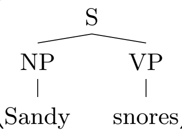
Probabilistic context-free grammars

- 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	Rule r	θ_r
S $\rightarrow$ NP VP	1.0		
NP $\rightarrow$ Sam	0.75	NP $\rightarrow$ Sandy	0.25
VP $\rightarrow$ barks	0.6	VP $\rightarrow$ snores	0.4

P (


) = 0.45

P (


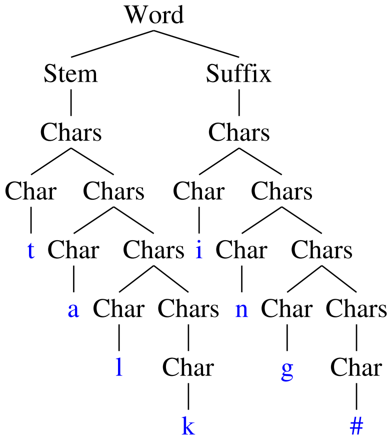
) = 0.1

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

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



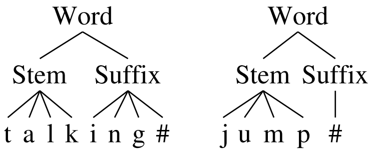
- Grammar's trees can represent any segmentation of words into stems and suffixes

$\Rightarrow$ 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

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



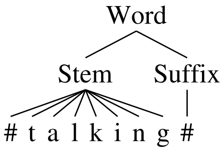
- A rule for each morpheme
  ⇒ “PCFG” can represent probability of each morpheme
- *Unbounded number of possible rules, so this is not a PCFG*
  - not a practical problem, as only a finite set of rules could possibly be used in any particular data set

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

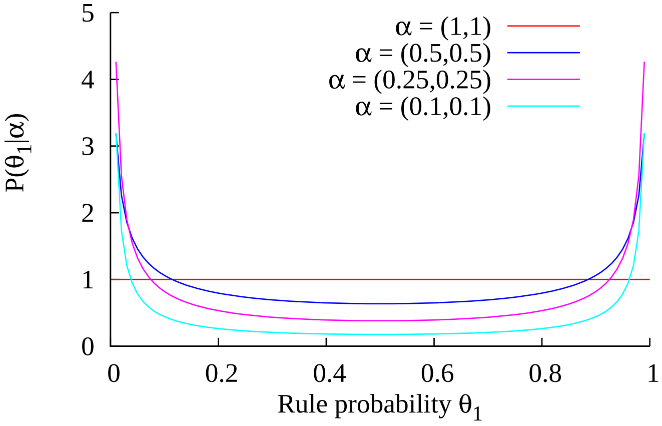
⇒ Saturated model is maximum likelihood estimate

- Maximum likelihood estimate does not find any suffixes



Forcing generalization 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 $\theta_{A\to\beta} \approx 0$
- Dirichlet prior with $\alpha_{A\to\beta} \approx 0$ prefers $\theta_{A\to\beta} \approx 0$



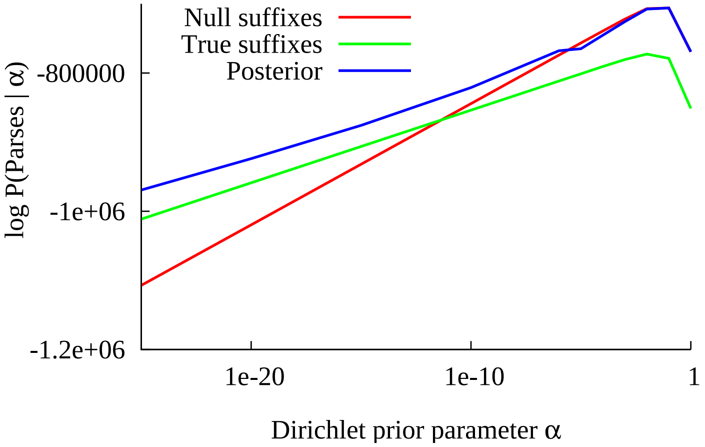
Morphological segmentation experiment

- Trained on orthographic verbs from U Penn. Wall Street Journal treebank
- Uniform Dirichlet prior prefers sparse solutions as $\alpha \to 0$
- 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	

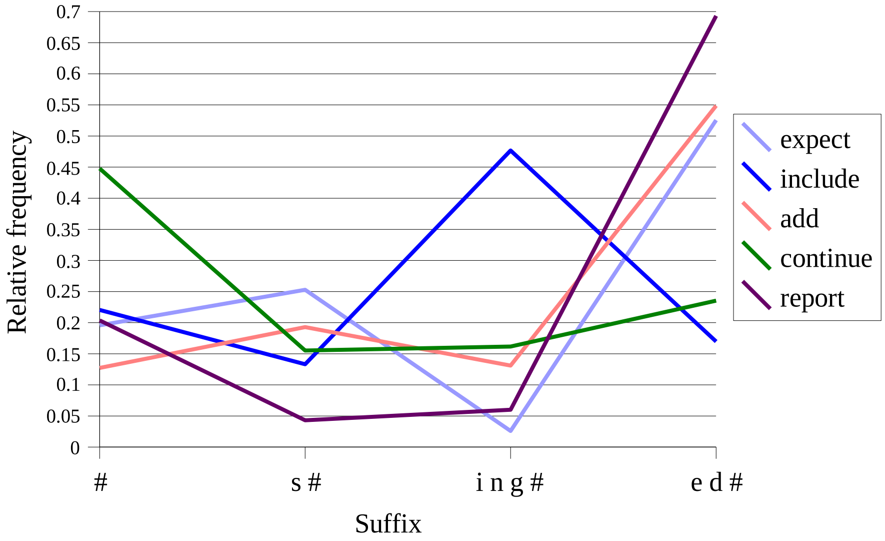
Log posterior for models on token data



- Correct solution is nowhere near as likely as posterior

⇒ 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

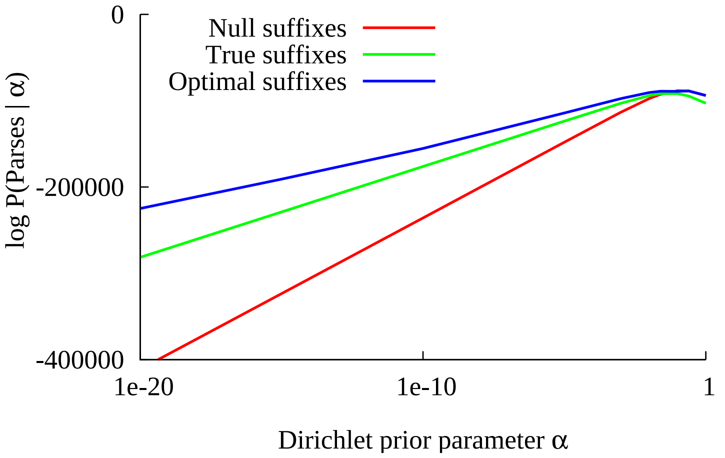
Data	=	“the cat chased the other cat”
Tokens	=	“the”, “cat”, “chased”, “the”, “other”, “cat”
Types	=	“the”, “cat”, “chased”, “other”

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

Outline

Dirichlet-Multinomials with many outcomes

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

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

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

$$\mathrm{P}(Z_{n+1} = k \mid \boldsymbol{z}, \alpha) \quad \propto \quad \begin{cases} n_k(\boldsymbol{z}) & \text{if } n_k(\boldsymbol{z}) > 0 \\[2ex] \alpha/m & \text{if } n_k(\boldsymbol{z}) = 0 \end{cases}$$

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

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

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:

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

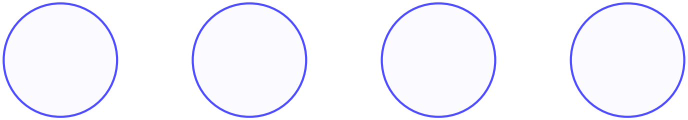
- Let $m \rightarrow \infty$

$$\begin{aligned} \mathrm{P}(Z_{n+1} = k \mid \boldsymbol{z}) &\;\propto\; n_k(\boldsymbol{z}) \text{ if } k \text{ appears in } \boldsymbol{z} \\ \mathrm{P}(Z_{n+1} \notin \boldsymbol{z} \mid \boldsymbol{z}) &\;\propto\; \alpha \end{aligned}$$

- 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)

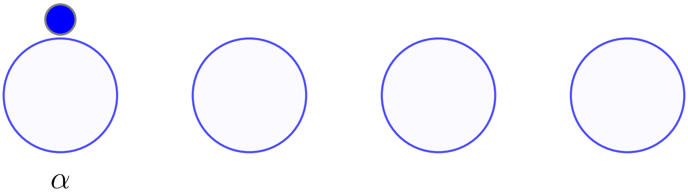


- Customer $\rightarrow$ table mapping $\boldsymbol{z} =$
- $\mathrm{P}(\boldsymbol{z}) = 1$

- 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)

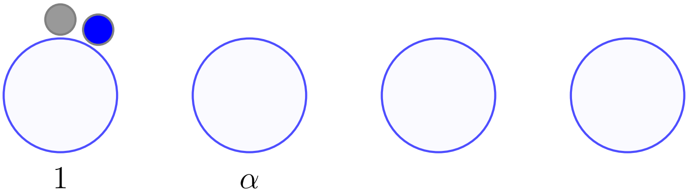


- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha$

- 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)

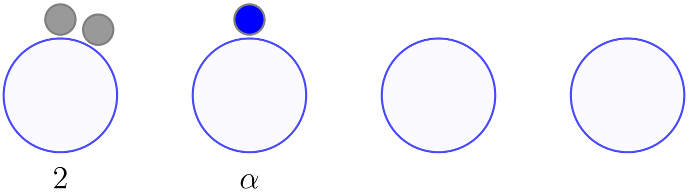


- Customer $\to$ table mapping $\boldsymbol{z} = 1, 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha)$

- 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)

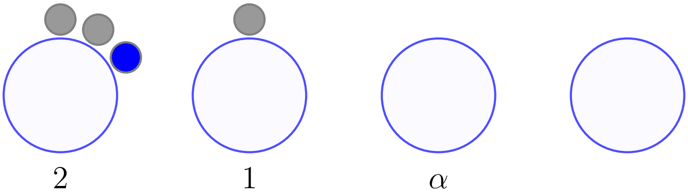


- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1, 2$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha) \times \alpha/(2+\alpha)$

- 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)

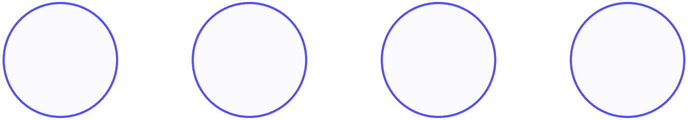


- Customer $\rightarrow$ table mapping $\boldsymbol{z} = 1, 1, 2, 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha) \times \alpha/(2+\alpha) \times 2/(3+\alpha)$

- Next customer chooses a table according to:

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

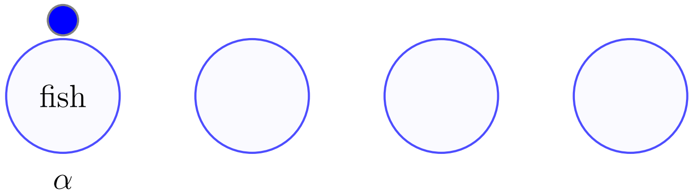
Labeled Chinese Restaurant Process (0)



- 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 $x_i = y_{z_i}$

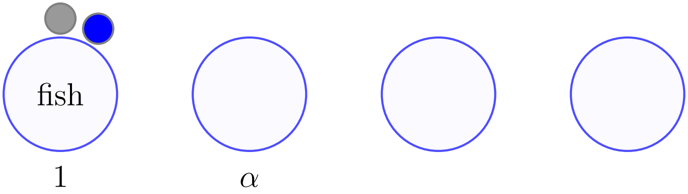
Labeled Chinese Restaurant Process (1)



- 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 $x_i = y_{z_i}$

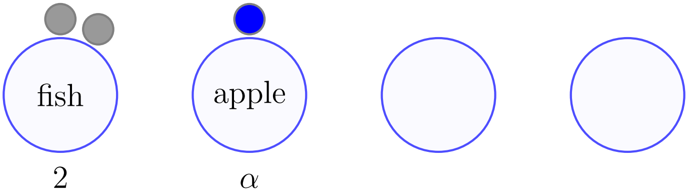
Labeled Chinese Restaurant Process (2)



- Table $\rightarrow$ label mapping $\boldsymbol{y}$ = fish
- Customer $\rightarrow$ 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 $x_i = y_{z_i}$

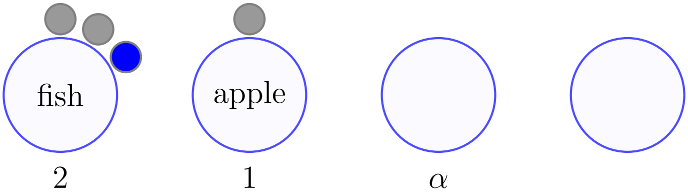
Labeled Chinese Restaurant Process (3)



- 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* 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., $\mathrm{S}_{35} \rightarrow \mathrm{NP}_{27}\ \mathrm{VP}_{17}$

    $\Rightarrow$ 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

Adaptor grammars: informal description

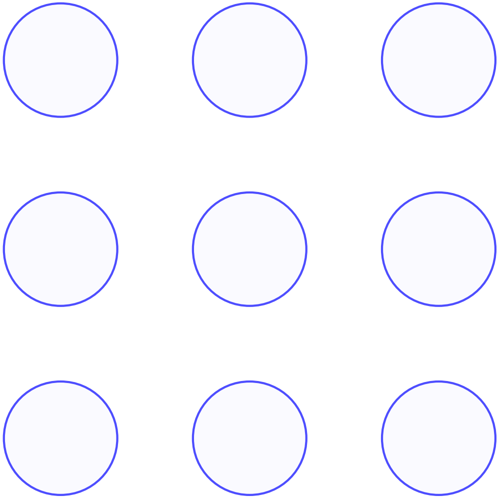
- The trees generated by an adaptor grammar are defined by CFG rules as in a CFG
- A subset of the nonterminals are *adapted*
- *Unadapted nonterminals* expand by picking a rule and recursively expanding its children, as in a PCFG
- *Adapted nonterminals* can expand in two ways:
  - by picking a rule 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 → Stem Suffix

Stem → Phoneme^{+}

Suffix → Phoneme*



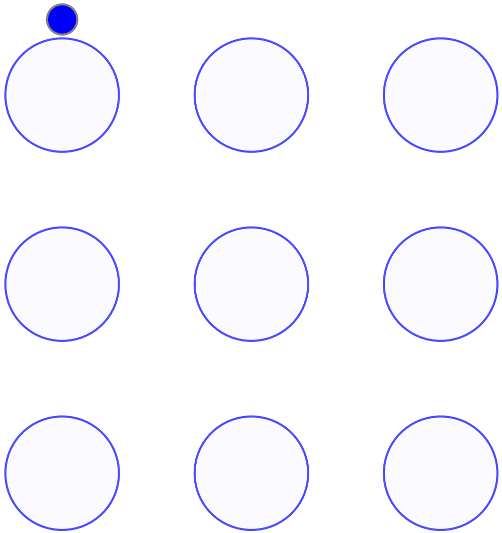
Generated words:

Adaptor grammar for stem-suffix morphology (1a)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



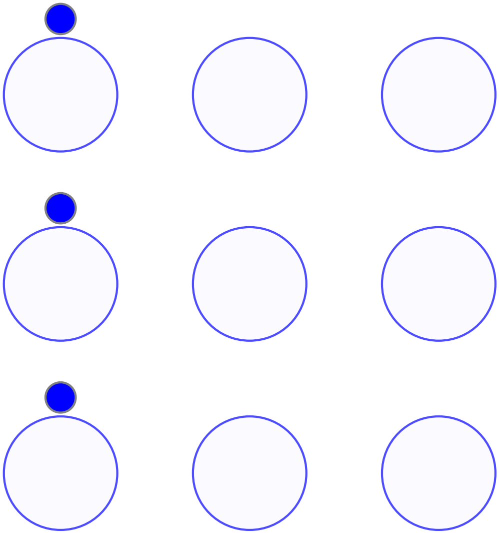
Generated words:

Adaptor grammar for stem-suffix morphology (1b)

Word → Stem Suffix

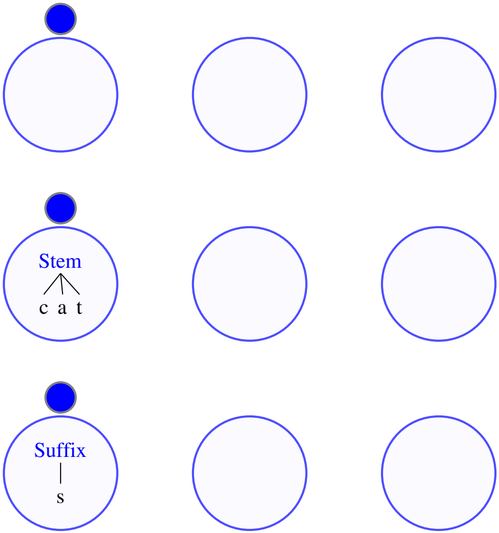
Stem → Phoneme^+

Suffix → $\text{Phoneme}^\star$



Generated words:

Adaptor grammar for stem-suffix morphology (1c)



Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*

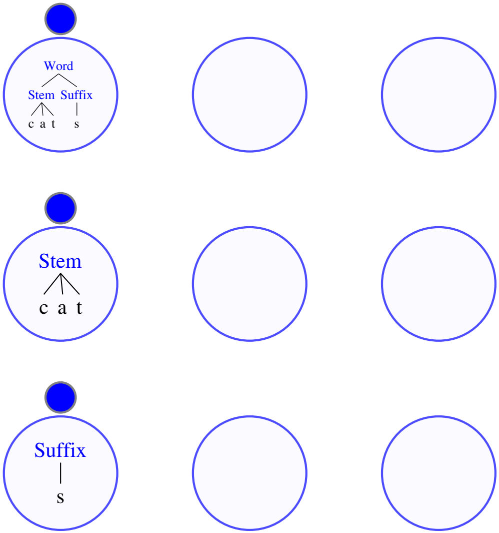
Generated words:

Adaptor grammar for stem-suffix morphology (1d)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



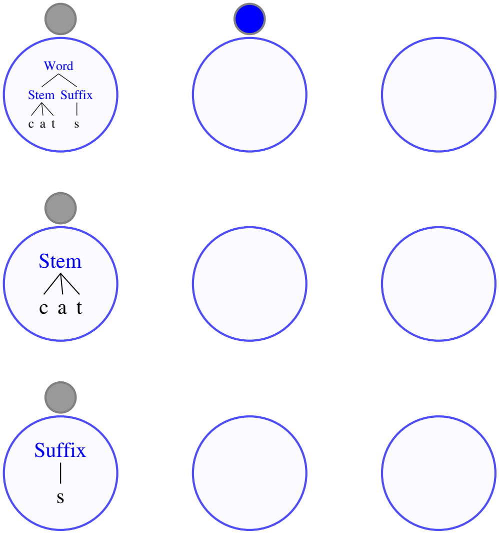
Generated words: cats

Adaptor grammar for stem-suffix morphology (2a)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



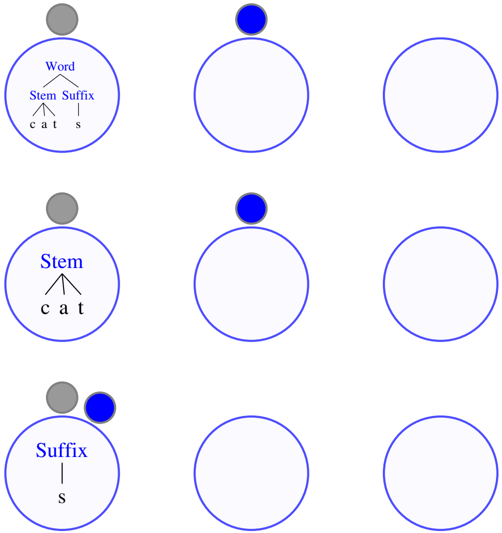
Generated words: cats

Adaptor grammar for stem-suffix morphology (2b)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



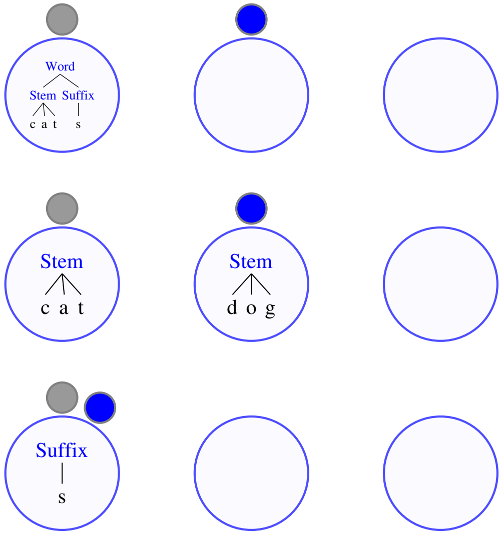
Generated words: cats

Adaptor grammar for stem-suffix morphology (2c)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



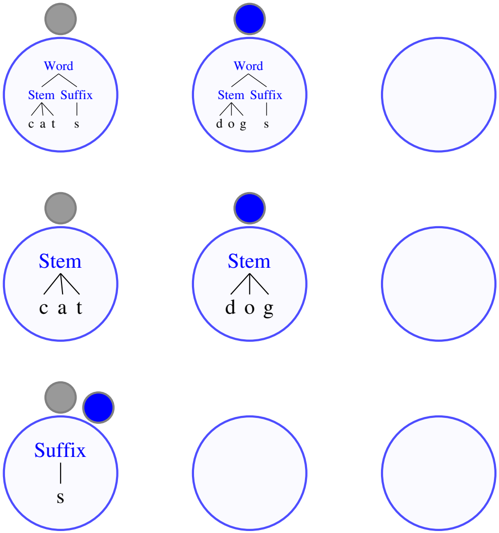
Generated words: cats

Adaptor grammar for stem-suffix morphology (2d)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



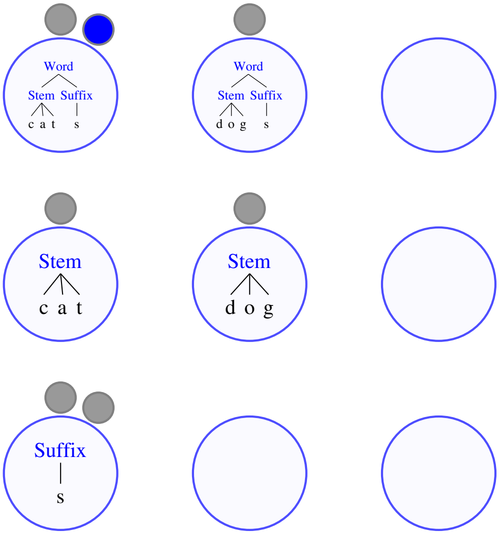
Generated words: cats, dogs

Adaptor grammar for stem-suffix morphology (3)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



Generated words: cats, dogs, cats

Adaptor grammars as generative processes

- 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 $A \to \beta$ with probability $\theta_{A \to \beta}$
- 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 $A \to \beta$ with probability proportional to $\alpha_A \theta_{A \to \beta}$

Properties of adaptor grammars

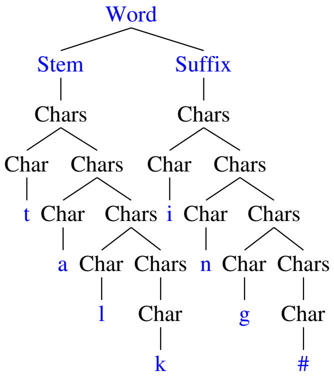
- Possible trees are generated by CFG rules
  but the probability of each adapted tree is learned separately
- Probability of adapted subtree τ is proportional to:
  - the number of times τ was seen before
    $\Rightarrow$ "rich get richer" dynamics (Zipf distributions)
  - plus α_A times prob. of generating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

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

- PCFG rule probabilities estimated *from table labels*
  $\Rightarrow$ effectively *learns from types*, not tokens
  $\Rightarrow$ 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
  $\Rightarrow$ higher in Bayesian hierarchy



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

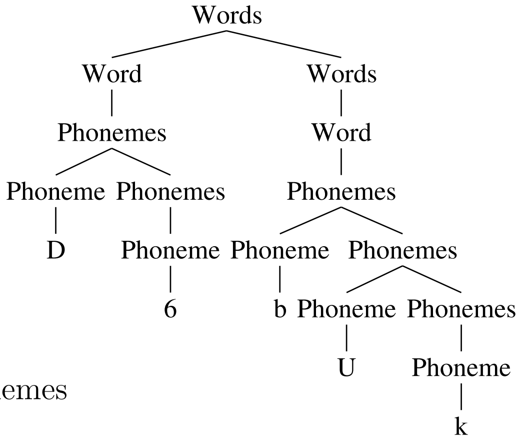
- Useful cues for word segmentation:
  - Phonotactics (Fleck)
  - Inter-word dependencies (Goldwater)

Word segmentation with PCFGs (1)

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

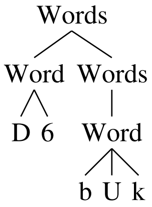
which abbreviates

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



Word segmentation with PCFGs (1)

Sentence → Word$^+$
Word → all possible phoneme strings

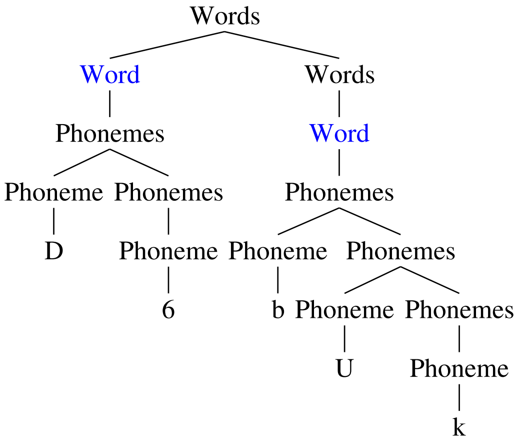


- But now there are an infinite number of PCFG rules!
  - once we see our (finite) training data, only finitely many are useful
  - ⇒ the set of parameters (rules) should be chosen based on training data

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

Adaptor grammar learnt from Brent corpus

- *Initial grammar*

1	Sentence → Word Sentence	1	Sentence → Word
1	Word → Phons		
1	Phons → Phon Phons	1	Phons → Phon
1	Phon → D	1	Phon → G
1	Phon → A	1	Phon → E

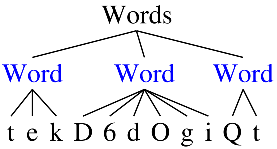
- *A grammar learnt from Brent corpus*

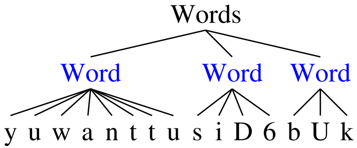
16625	Sentence → Word Sentence	9791	Sentence → Word
1	Word → Phons		
4962	Phons → Phon Phons	1575	Phons → Phon
134	Phon → D	41	Phon → G
180	Phon → A	152	Phon → E
460	Word → (Phons (Phon *y*) (Phons (Phon *u*)))		
446	Word → (Phons (Phon *w*) (Phons (Phon *A*) (Phons (Phon *t*))))		
374	Word → (Phons (Phon *D*) (Phons (Phon *6*)))		
372	Word → (Phons (Phon *&*) (Phons (Phon *n*) (Phons (Phon *d*))))		

Words (unigram model)

$$\text{Sentence} \to \text{Word}^{+} \qquad \text{Word} \to \text{Phoneme}^{+}$$

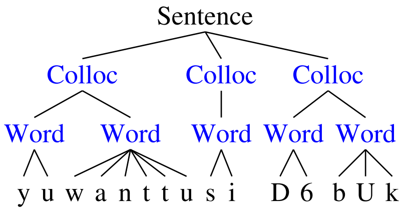
- Unigram word segmentation model assumes each word is generated independently
- But there are strong inter-word dependencies (collocations)
- Unigram model can only capture such dependencies by analyzing collocations as words (Goldwater 2006)





Collocations ⇒ Words

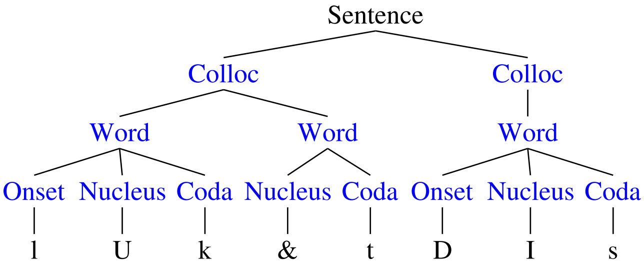
$$\begin{aligned}&\text{Sentence} \to \text{Colloc}^+\\ &\underline{\text{Colloc}} \to \text{Word}^+\\ &\underline{\text{Word}} \to \text{Phon}^+\end{aligned}$$



- 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; ≈ Goldwater's bigram model)

Collocations ⇒ Words ⇒ Syllables

Sentence → Colloc^+	Colloc → Word^+
Word → Syllable	Word → Syllable Syllable
Word → Syllable Syllable Syllable	Syllable → (Onset) Rhyme
Onset → Consonant^+	Rhyme → Nucleus (Coda)
Nucleus → Vowel^+	Coda → Consonant^+

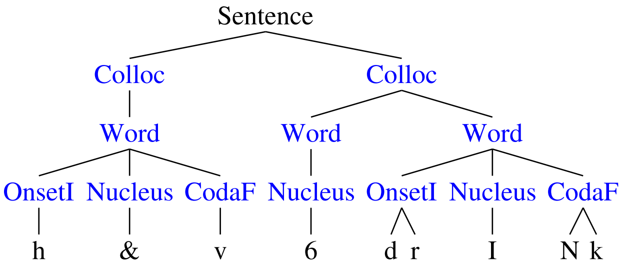


- With no supra-word generalizations, f-score = 68%
- With 2 Collocation levels, f-score = 82%

Distinguishing internal onsets/codas helps

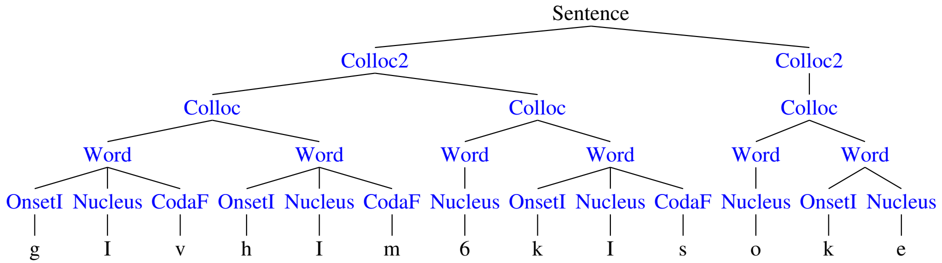
Sentence → Colloc$^+$
Word → SyllableIF
Word → SyllableI Syllable SyllableF
OnsetI → Consonant$^+$
Nucleus → Vowel$^+$

Colloc → Word$^+$
Word → SyllableI SyllableF
SyllableIF → (OnsetI) Rhyme
RhymeF → Nucleus (CodaF)
CodaF → Consonant$^+$



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

Collocations^2 ⇒ Words ⇒ Syllables

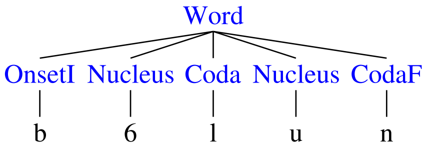

Sentence
Colloc2
Colloc2
Colloc
Colloc
Colloc
Word
Word
Word
Word
Word
Word
OnsetI Nucleus CodaF OnsetI Nucleus CodaF Nucleus OnsetI Nucleus CodaF Nucleus OnsetI Nucleus
g I v h I m 6 k I s o k e

Syllabification learnt by adaptor grammars

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

$$1 \quad \text{Syllable} \rightarrow \text{Onset Rhyme} \qquad 1 \quad \text{Syllable} \rightarrow \text{Rhyme}$$

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

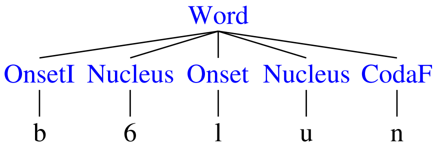


- Syllabification accuracy is relatively poor
  Syllabification given true word boundaries: f-score = 83%
  Syllabification learning word boundaries: f-score = 74%

Preferring Onsets improves syllabification

2 Syllable → Onset Rhyme 1 Syllable → Rhyme

- Changing the prior to prefer word-internal Syllables with Onsets dramatically improves segmentation accuracy
- "Rich get richer" property of Chinese Restaurant Processes
  ⇒ 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

$$\begin{array}{ll} \text{Onset} \rightarrow \text{Onset}_{\text{Stop}} & \text{Onset} \rightarrow \text{Onset}_{\text{Fricative}} \\ \text{Onset}_{\text{Stop}} \rightarrow \text{Stop} & \text{Onset}_{\text{Stop}} \rightarrow \text{Stop}\ \text{Onset}_{\text{Fricative}} \\ \text{Stop} \rightarrow \text{p} & \text{Stop} \rightarrow \text{t} \end{array}$$

- 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., $\text{Onset}_{\text{Stop}} \rightarrow \text{Stop}\ \text{Onset}_{\text{Fricative}}$) to transitions that aren't (e.g., $\text{Onset}_{\text{Fricative}} \rightarrow \text{Fricative}\ \text{Onset}_{\text{Stop}}$)
- 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*
- Adding morphology does not seem to help

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$^+$	Unordered $\rightarrow$ Word$^+$
NP $\rightarrow$ (A0) (A1) ... (A6)	NP $\rightarrow$ (B0) (B1) ... (B6)
A0 $\rightarrow$ Word$^+$	B0 $\rightarrow$ Word$^+$
...	...
A6 $\rightarrow$ 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

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
  - $\Rightarrow$ 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

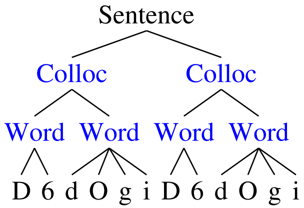
- Intuition: same as simple incremental algorithm, but re-use sentences in training data
    - Assign (random) parse trees to each sentence, and compute rule and subtree counts
    - Repeat forever:
        - pick a sentence (and corresponding parse) at random
        - deduct the counts for the sentence's 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 $G'(\boldsymbol{t}_{-j})$ consist of:
  - rules $A \rightarrow \beta$ from base PCFG: $\theta'_{A\rightarrow\beta} \;\propto\; \alpha_A \theta_{A\rightarrow\beta}$
  - A rule $A \rightarrow \textsc{Yield}(\tau)$ for each table τ in A's restaurant: $\theta'_{A\rightarrow\textsc{Yield}(\tau)} \;\propto\; n_\tau$, the number of customers at table τ
- Parses of $G'(\boldsymbol{t}_{-j})$ 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
  $\Rightarrow$ 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

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

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 δ_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 $A \in N$, α_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:

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

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

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