Probabilistic Models for Computational Linguistics

Mark Johnson

Macquarie University

Machine Learning summer school 2010

Outline

The big ideas

- Probabilistic generative models
- The *noisy channel* model, and recovering hidden structure
- The *Expectation Maximisation* (EM) algorithm
- *Hidden Markov Models* (HMMs)
- *Probabilistic Context-Free Grammars* (PCFGs)
- *Maximum Entropy* (MaxEnt) models
- *Non-parametric Bayesian inference* (Chinese restaurant processes)

MACQUARIE UNIVERSITY

Brief Introduction to Discrete Probability

- *Sample space* Ω: set of possible events
- *Probability* $\mathrm{P}(x)$ of event $x \in \Omega$
  - $0 \leq \mathrm{P}(x) \leq 1$ for all $x \in \Omega$
  - $\sum_{x\in\Omega} \mathrm{P}(x) = 1$
- A *random variable* is a function on Ω
  - If A is a random variable and a is one of its values,

$$\mathrm{P}(A = a) \quad = \quad \sum_{\substack{x\in\Omega \\ A(x)=a}} \mathrm{P}(x)$$

  i.e., $\mathrm{P}(A = a)$ is sum of probability of all events x where $A(x) = a$.

- Example: $\Omega = \{\mathrm{Sam}, \mathrm{Sandy}, \mathrm{likes}\}$
  $\mathrm{P}(\mathrm{Sam}) = 0.5, \mathrm{P}(\mathrm{Sandy}) = 0.1, \mathrm{P}(\mathrm{likes}) = 0.4$
  $\mathrm{PoS}(\mathrm{Sam}) = \mathrm{n}, \mathrm{PoS}(\mathrm{Sandy}) = \mathrm{n}, \mathrm{PoS}(\mathrm{likes}) = \mathrm{v}$
  $\mathrm{P}(\mathrm{PoS} = \mathrm{n}) = 0.6, \mathrm{P}(\mathrm{PoS} = \mathrm{v}) = 0.4$

Joint and conditional probabilities

- *Joint probability* of two random variables A and B is:

$$\mathrm{P}(A = a, B = b) \quad = \quad \sum_{\substack{x \in \Omega \\ A(x)=a \\ B(x)=b}} \mathrm{P}(x)$$

  is sum of probability of all events x where $A(x) = a$ and $B(x) = b$

- *Conditional probability* of A *given* B is:

$$\mathrm{P}(A = a \mid B = b) \quad = \quad \frac{\mathrm{P}(A = a, B = b)}{\mathrm{P}(B = b)}$$

- Example: $\Omega = \{\mathrm{Sam}, \mathrm{Sandy}, \mathrm{likes}\}$
  $\mathrm{P}(\mathrm{Sam}) = 0.5, \mathrm{P}(\mathrm{Sandy}) = 0.1, \mathrm{P}(\mathrm{likes}) = 0.4$
  $\mathrm{Len}(x) =$ number of chars in x,
  $\mathrm{Pos}(\mathrm{Sam}) = \mathrm{PoS}(\mathrm{Sandy}) = \mathrm{n}, \mathrm{PoS}(\mathrm{likes}) = \mathrm{v}$

$$\mathrm{P}(\mathrm{PoS} = \mathrm{n} \mid \mathrm{Len} = 5) \quad = \quad \frac{\mathrm{P}(\mathrm{PoS} = \mathrm{n}, \mathrm{Len} = 5)}{\mathrm{P}(\mathrm{Len} = 5)} \quad = \quad 0.2$$

MACQUARIE UNIVERSITY

Bayes rule

- For any random variables A, B:

$$\mathrm{P}(A, B) \;=\; \mathrm{P}(A \mid B)\,\mathrm{P}(B) \;=\; \mathrm{P}(B \mid A)\,\mathrm{P}(A)$$

so:

$$\mathrm{P}(A \mid B) \;=\; \frac{\mathrm{P}(B \mid A)\,\mathrm{P}(A)}{\mathrm{P}(B)}$$

- Bayes rule is useful for *inverting conditional probability distributions*

MACQUARIE UNIVERSITY

The Noisy Channel model and Bayes rule

- The *Noisy Channel model* uses Bayes rule to invert probability distributions
- Example: Speech recognition maps *acoustic inputs* A to *texts* T
- Probabilistic formulation:

$$\begin{aligned} T^{\star}(A) &= \operatorname*{argmax}_{T} \mathrm{P}(T \mid A) \\ &= \operatorname*{argmax}_{T} \frac{\mathrm{P}(A \mid T)\,\mathrm{P}(T)}{\mathrm{P}(A)} \\ &= \operatorname*{argmax}_{T} \mathrm{P}(A \mid T)\,\mathrm{P}(T) \end{aligned}$$

- $\mathrm{P}(T)$ is called the *source model*
- $\mathrm{P}(A \mid T)$ is called the *channel model*
- Other applications of noisy channel models:
  - machine translation
  - language reconstruction in historical linguistics
  - detecting and correcting disfluencies in speech recognition

MACQUARIE UNIVERSITY

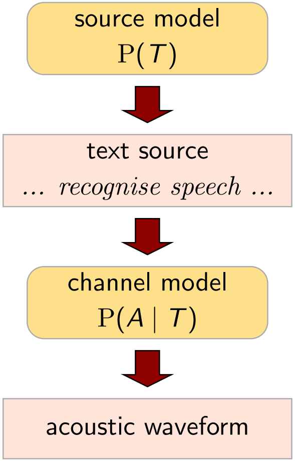
The Noisy Channel model as a generative process

- The *source model* generates *text strings* T

$$\mathrm{P}(\mathrm{T}=\text{“\dots recognise speech \dots”}) > \mathrm{P}(T = \text{“\dots wreck a nice beach \dots”})$$

- The *channel model* maps text T to an *acoustic waveform* A
- Computing with the noisy channel model uses Bayes rule to compute

$$\mathrm{P}(T \mid A) = \frac{\mathrm{P}(A \mid T)\,\mathrm{P}(T)}{\mathrm{P}(A)}$$



Outline

Language modelling overview

- Goal of language modelling: distinguish more likely text or speech from less likely text or speech
- A key component of many NLP systems
    - Speech recognition: e.g., "recognise speech" vs "wreck a nice beach"
    - Machine translation
- *n*-gram models model text in terms of overlapping *n*-word sequences
    - easy to train from billions of words of text
    - produce quite good results
- They also introduce key notions of probability and statistics

What is a language model?

- A sentence or a document is a *sequence of words* $\boldsymbol{w} = (w_1, \ldots, w_n)$, where each $w_i \in \mathcal{W}$ (i.e., $\mathcal{W}$ is the vocabulary)
- Goal: find an accurate probability distribution $\mathrm{P}(\boldsymbol{w})$ over word sequences
- Method: *estimate* $\mathrm{P}(\boldsymbol{w})$ using *statistics* collected from a *corpus* of sentences
  - a *statistic* is a function of the data
- Questions:
  - which statistics should we collect?
  - how do we use them to estimate $\mathrm{P}(\boldsymbol{w})$?
- Why is this hard?

MACQUARIE UNIVERSITY

“Bag of words” (unigram) models

- “Bag of words” assumption: for all positions i, j in a sentence

$$\mathrm{P}(W_i = w) \;\; = \mathrm{P}(W_j = w)$$

- To generate a sentence $\boldsymbol{w} = (w_1, \ldots, w_n)$:
  1. generate a *sentence length* n from $\mathrm{P}(N)$
  2. generate word i from a word distribution $\mathrm{P}(W_i)$

$$\mathrm{P}(\boldsymbol{W} = (w_1, \ldots, w_n)) \;\; = \;\; \mathrm{P}(N = n) \prod_{i=1}^{n} \mathrm{P}(W_i = w_i)$$

- A unigram model lets us define a distribution over sentences in terms of distributions over sentence lengths and distributions over words

Estimating $\mathrm{P}(W)$ from a corpus

- Informal idea:
  - Given a corpus D, count how often each word w appears
  - Set $\mathrm{P}(W = w)$ to the fraction of times w appears in D
- Formalise as a *parameter estimation problem*
  - Introduce parameters

$$\mathrm{P}(W_i = w) \quad = \quad \varphi_w \text{ for all } w \in \mathcal{W} \text{ and } i = 1, 2, \ldots$$

    so $\boldsymbol{\varphi}$ is a *vector* indexed by *word types* in $\mathcal{W}$
  - Provide an *estimator* $\widehat{\boldsymbol{\varphi}}(D)$ for the parameters from data D

$$\begin{aligned} \widehat{\varphi}_w &= \frac{n_w(D)}{n(D)}, \text{ where:} \\ n_w(D) &= \text{number of times } w \text{ appears in } D, \text{ and} \\ n_{\cdot}(D) &= \sum_{w \in \mathcal{W}} n_w(D) \;=\; \text{number of words in } D \end{aligned}$$

Maximum likelihood estimation

- *Maximum likelihood principle:*
  Choose model parameters to make data *as likely as possible*
- *Likelihood function:* probability of data *as function of model parameters*
- If $D = (w_1, \ldots, w_n)$ (i.e., one long sentence)

$$L_D(\boldsymbol{\varphi}) \;=\; \mathrm{P}_{\varphi}(D) \;=\; \mathrm{P}(N = n) \prod_{i=1}^{n} \varphi_{w_i}$$

- Maximum Likelihood Estimate (MLE):

$$\widehat{\boldsymbol{\varphi}} \;=\; \operatorname*{argmax}_{\boldsymbol{\varphi}} L_D(\boldsymbol{\varphi})$$

- Fact: MLE is relative frequency, i.e.,

$$\widehat{\boldsymbol{\varphi}}_w \;=\; \frac{n_w(D)}{n_{\cdot}(D)}$$

MACQUARIE UNIVERSITY

Smoothing the MLE $\widehat{\varphi}$

- Suppose $\boldsymbol{w}$ contains an *unknown word* w, i.e., $n_w(D) = 0$
  $\Rightarrow\ \widehat{\varphi}_w = 0$
  $\Rightarrow\ \mathrm{P}_{\boldsymbol{\varphi}}(\boldsymbol{w}) = 0$

$\Rightarrow$ Use a different estimator, e.g., *Bayesian MAP estimator*

$$\begin{aligned} \tilde{\varphi}_w &= \frac{n_w(D) + 1}{n_\cdot(D) + m}, \text{ where:} \\ m &= |\mathcal{W}| = \text{size of vocabulary} \end{aligned}$$

- The Bayesian MAP estimator *smooths* the MLE

More on unknown words

- In many applications you'll often encounter unknown words, no matter how big training data D is
- One approach: replace all words not seen (say) 5 times in D with a special symbol, say $\star\mathrm{UNK}\star$
- Problem: for many infrequent words w, usually $\varphi_{\star\mathrm{UNK}\star} > \varphi_w$
- More sophisticated models of unknown words make a big difference
- "Lazy statistical modeller's morphology"
  - Multiple kinds of "unknown words"
  - classify unknown words by
    - capitalisation, or
    - last three letters

    e.g., $\mathrm{Melbourne} \mapsto \star\mathrm{UNK}_{\mathrm{CAP}}\star$, $\mathrm{walking} \mapsto \star\mathrm{UNK}_{\mathrm{ing}}\star$

Bigram language model

- A *bigram language model* captures dependencies between adjacent words
- Conditional decomposition: (exact)

$$\begin{aligned}
\mathrm{P}(w_1, \ldots, w_n) &= \mathrm{P}(w_1, \ldots, w_{n-1})\, \mathrm{P}(w_n \mid w_1, \ldots, w_{n-1}) \\
&= \mathrm{P}(w_1)\, \mathrm{P}(w_2 \mid w_1)\, \mathrm{P}(w_3 \mid w_1, w_2)\, \ldots \\
&\quad\ \mathrm{P}(w_n \mid w_1, \ldots, w_{n-1})
\end{aligned}$$

- *Markov:* only last word matters (approximation)

$$\begin{aligned}
\mathrm{P}(w_i \mid w_1, \ldots, w_{i-1}) &= \mathrm{P}(w_i \mid w_{i-1}), \text{ so:} \\
\mathrm{P}(w_1, \ldots, w_n) &= \mathrm{P}(w_1)\, \mathrm{P}(w_2 \mid w_1)\, \mathrm{P}(w_3 \mid w_2)\, \ldots\, \mathrm{P}(w_n \mid w_{n-1}
\end{aligned}$$

- *Stationarity:* position doesn't matter (approximation)

$$\mathrm{P}(w_i \mid w_{i-1}) = \mathrm{P}(w_j \mid w_{j-1}) \text{ for all } i, j$$

MACQUARIE UNIVERSITY

Sentence boundary markers

- Instead of generating $w_1, \ldots, w_n$, generate the infinite sequence

$$\triangleright, w_1, \ldots, w_n, \triangleleft, \triangleleft, \ldots$$

where $w_0 = \triangleright$ is the *beginning of sentence marker*
and $w_{n+1} = \triangleleft$ is the *end of sentence marker*

$\Rightarrow$ Everything is a bigram dependency, e.g., $\mathrm{P}(w_1) = \mathrm{P}(w_1 \mid \triangleright)$; $\mathrm{P}(\triangleleft \mid \triangleleft) = 1$

- Bigram language model:

$$\begin{aligned} \mathrm{P}(w_1, \ldots, w_n) &= \prod_{i=1}^{n+1} \mathrm{P}(w_i \mid w_{i-1}) \\ &= \prod_{i=1}^{n+1} \theta_{w_i, w_{i-1}} \end{aligned}$$

where $\theta_{w,w'} = \mathrm{P}(W_i = w \mid W_{i-1} = w')$ (stationarity)

- $\boldsymbol{\theta}$ is a matrix of transition probabilities indexed by $\mathcal{W}$.

Bigram example

- Bigram parameters:

$\boldsymbol{\theta} =$

$w_i \backslash w_{i-1}$	$\triangleright$	likes	Sam	Sandy
likes	0.1	0.1	0.2	0.4
Sam	0.5	0.2	0.1	0.1
Sandy	0.4	0.6	0.1	0.1
$\triangleleft$	0	0.1	0.6	0.4

- Bigram model estimates:

$$\begin{aligned} \mathrm{P}(\triangleright, \mathrm{Sam}, \triangleleft) &= \theta_{\mathrm{Sam},\triangleright}\, \theta_{\triangleleft,\mathrm{Sam}} \\ &= 0.5 \times 0.6 \\ \mathrm{P}(\triangleright, \mathrm{Sandy}, \mathrm{likes}, \mathrm{Sam}, \triangleleft) &= \theta_{\mathrm{Sandy},\triangleright}\, \theta_{\mathrm{likes},\mathrm{Sandy}}\, \theta_{\mathrm{Sam},\mathrm{likes}}\, \theta_{\triangleleft,\mathrm{Sam}} \\ &= 0.4 \times 0.4 \times 0.2 \times 0.6 \end{aligned}$$

MACQUARIE UNIVERSITY

Estimating bigram model parameters $\boldsymbol{\theta}$

- Maximum Likelihood Estimate from corpus D:

$$\begin{aligned}\widehat{\theta}_{w,w'} &= \frac{n_{w,w'}(D)}{n_{\cdot,w'}(D)}, \text{ where:}\\ n_{w,w'}(D) &= \text{number of times } w \text{ follows } w' \text{ in } D\text{, and}\\ n_{\cdot,w'}(D) &= \text{number of times } w' \text{ follows anything in } D\end{aligned}$$

- Data sparsity is even more problematic for bigram models
- Many methods for *smoothing using a unigram model* $\widehat{\varphi}$, e.g.:

$$\tilde{\theta}_{w,w'} = \lambda\widehat{\varphi}_w + (1-\lambda)\widehat{\theta}_{w,w'}$$

- Choose *interpolation parameter* λ to maximize likelihood of a *heldout corpus*

MACQUARIE UNIVERSITY

Further work in language modelling

- Standard to work with 3-gram to 5-gram models estimated from billions of words of text
- Smoothing is essential, and the method used makes a big difference
- Wide variety of language models have been investigated
  - *Trigger language models* try to track the change in vocabulary within a document
  - *Syntactic (i.e., parsing-based) language models* typically outperform *n*-grams when *trained on same sized corpora*

Outline

MACQUARIE UNIVERSITY

Machine translation overview

- A *machine translation system* automatically translates text or speech from one language to another
- Uses the noisy channel to decompose the problem into language model and translation model
- Introduces the *Expectation Maximisation* (EM) algorithm for learning from *hidden data*
  - translation model maps *words to words* or *phrases to phrases*
  - most naturally learnt from *word-aligned* parallel translations
  - but virtually all training data is *sentence-aligned*

  $\Rightarrow$ word alignments are *hidden*
- EM learns *word alignments* and *word translation probabilities* from sentence-aligned data

Noisy channel model for translation

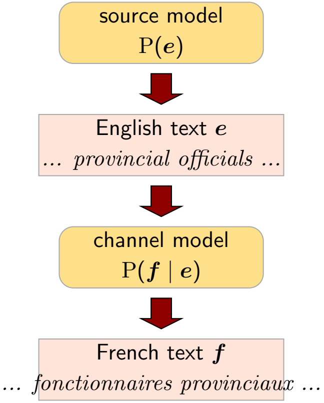
- Goal: translate a sentence $\boldsymbol{f}$ (e.g., "foreign", "French") into a sentence $\boldsymbol{e}$ ("English")

$$E^{\star}(\boldsymbol{f}) = \underset{\boldsymbol{e}}{\operatorname{argmax}} \mathrm{P}(\boldsymbol{e} \mid \boldsymbol{f})$$

- Noisy channel model (i.e., Bayes inversion)

$$\underset{\boldsymbol{e}}{\operatorname{argmax}} \mathrm{P}(\boldsymbol{e} \mid \boldsymbol{f}) = \underset{\boldsymbol{e}}{\operatorname{argmax}} \mathrm{P}(\boldsymbol{e})\, \mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})$$

- $\mathrm{P}(\boldsymbol{e})$ is a *language model*, typically trained on *billions* of words of monolingual text
- $\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})$ is a *channel model*, typically trained on *millions* of words of parallel text



Sentence-aligned parallel corpus (Canadian Hansards)

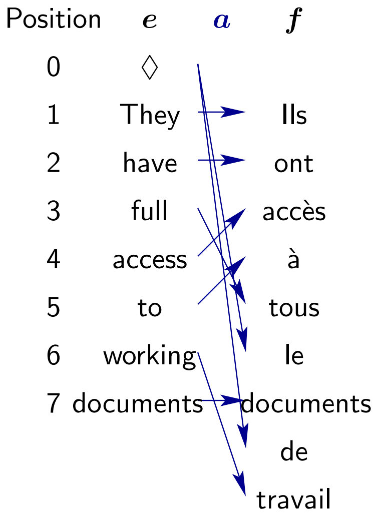
- *English:*
  provincial officials are consulted through conference calls and negotiated debriefings .
  they have full access to working documents .
  consultations have also taken place with groups representing specific sectors of the Canadian economy , including culture , energy , mining , telecommunications and agrifood .
- *French:*
  les fonctionnaires provinciaux sont consultés par appels conférence et comptes rendus .
  ils ont accès à tous les documents de travail .
  les consultations ont aussi eu lieu avec de les groupes représentant de les secteurs précis de le économie canadienne , y compris la culture , le énergie , les mines , les télécommunications et le agro - alimentaire .

Channel models for machine translation

- Overview of machine translation channel model:
  - Analyse the English input $\boldsymbol{e}$ into a sequence of words or phrases
  - Replace each English word or phrase with its French translation
  - Reorder the translated words or phrases to produce French output $\boldsymbol{f}$
- Translation units and reorderings can be identified in many ways
  - here the translation units are *words*
  - initially we will assume *every reordering is equally likely*
  - but it's easy to use our methods to learn *string position based reordering*

The IBM 1 translation model $\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})$

- A *word alignment* $\boldsymbol{a}$ pairs each French word f_k to a single English word e_{a_k}.
- One English word may be aligned with several French words
- We add a special "null word" $\Diamond$ to e to pair French words with no English counterpart
- To generate $\boldsymbol{f}$ from $\boldsymbol{e}$:
  - Pick an alignment $\boldsymbol{a}$
  - For each k, generate f_k from e_{a_k} according to $\mathrm{P}(f_k \mid e_{a_k})$



$\boldsymbol{a} = (1, 2, 4, 5, 3, 0, 7, 06)$

An alternative representation of alignments

	They	have	full	access	to	working	documents
Ils	■						
ont		■					
accès				■			
à					■		
tous			■				
le							
documents							■
de							
travail						■	

MACQUARIE
UNIVERSITY

Mathematical formulation of IBM model 1

- Simple generative model:

$$\begin{aligned}
\mathrm{P}(\boldsymbol{a}, \boldsymbol{f} \mid \boldsymbol{e}) &= \mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e})\, \mathrm{P}(\boldsymbol{f} \mid \boldsymbol{a}, \boldsymbol{e}) \\
&= \mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}) \prod_{k=1}^{|\boldsymbol{a}|} \mathrm{P}(f_k \mid e_{a_k}), \text{ so:} \\
\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e}) &= \sum_{\boldsymbol{a}} \mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}) \prod_{k=1}^{|\boldsymbol{a}|} \mathrm{P}(f_k \mid e_{a_k})
\end{aligned}$$

- In our simple generative model, $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e})$ is a *uniform distribution* over alignments given the French sentence length $|\boldsymbol{f}|$
- Even so, $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$ is *not uniform!*

$$\begin{aligned}
\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f}) &= \frac{\mathrm{P}(\boldsymbol{a}, \boldsymbol{f} \mid \boldsymbol{e})}{\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})} \\
&\propto \prod_{k=1}^{|\boldsymbol{a}|} \mathrm{P}(f_k \mid e_{a_k})
\end{aligned}$$

MACQUARIE UNIVERSITY

Learning translation probabilities from a word-aligned corpus

- Goal: learn word translation probabilities $\mathrm{P}(f \mid e)$
- Easy with a *word-aligned parallel corpus* $D = (\boldsymbol{e}, \boldsymbol{f}, \boldsymbol{a})$
- Let $\mathrm{P}(f \mid e) = \tau_{f,e}$. Then e.g., the MLE is:

$$\begin{aligned}
\widehat{\tau}_{f,e} &= \frac{n_{f,e}(\boldsymbol{a})}{n_{\cdot,e}(\boldsymbol{a})}, \text{ where:} \\
n_{f,e}(\boldsymbol{a}) &= \text{the number of times } f \text{ is aligned to } e \text{ in } D \text{, and} \\
n_{\cdot,e}(\boldsymbol{a}) &= \sum_{f} n_{f,e}(\boldsymbol{a}) \\
&= \text{the number of times } e \text{ is aligned to anything in } D
\end{aligned}$$

- But word-aligned corpora are very expensive
  can we learn translation probabilities from sentence-aligned corpora instead?

MACQUARIE UNIVERSITY

The Expectation Maximisation Algorithm

- EM is useful for problems
  - with *hidden variables* (variables whose values are unknown, e.g., alignments $\boldsymbol{a}$), and
  - where estimation would be easy if those variables' values were known
- EM is an *iterative hill-climbing algorithm*
  - every iteration is *guaranteed* not to decrease the likelihood of data
  - but EM can "get stuck" in local maxima
- *I*nformal description of EM algorithm:
  - initialise parameters $\boldsymbol{\tau}^{(0)}$ somehow (e.g., randomly)
  - repeat for $t = 0, 1, \ldots,$ until convergence:
    - *E-step:* Use current parameter estimate $\boldsymbol{\tau}^{(t)}$ to "fill in" hidden variable $\boldsymbol{a}$ in training data, producing "expected data" $(\boldsymbol{a}, \boldsymbol{e}, \boldsymbol{f})$ with "expected count" $\mathrm{P}_{\boldsymbol{\tau}^{(t)}}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$
    - *M-step:* Estimate parameters $\boldsymbol{\tau}^{(t+1)}$ from expected data constructed in E-step

The Expectation Maximisation Algorithm for Model 1

- Too many possible alignments **a** to enumerate expected data $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$
  - but all we need to estimate $\boldsymbol{\tau}^{(t+1)}$ are the *sufficient statistics* $n_{f,e}$
- High-level description of EM algorithm:

  Initialise model parameters $\boldsymbol{\tau}^{(0)}$ somehow (e.g., randomly)
  Repeat for $t = 0, 1, \ldots$ until convergence:
  - *E-step:* Compute *expected value* of sufficient statistics

$$\mathrm{E}_{\boldsymbol{\tau}^{(t)}}[n_{f,e}] \quad = \quad \sum_{\boldsymbol{a}} n_{f,e}(\boldsymbol{a}) \, \mathrm{P}_{\boldsymbol{\tau}^{(t)}}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$$

  - *M-step:* Compute $\boldsymbol{\tau}^{(t+1)}$ from $\mathrm{E}[n_{f,e}]$

$$\tau_{f,e}^{(t+1)} \quad = \quad \frac{\mathrm{E}_{\boldsymbol{\tau}^{(t)}}[n_{f,e}(\boldsymbol{a})]}{\mathrm{E}_{\boldsymbol{\tau}^{(t)}}[n_{\cdot,e}(\boldsymbol{a})]}$$

Expected values of translation counts

- If g is a real-valued random variable, its *expected value* $\mathrm{E}[g]$ is:

$$\mathrm{E}[g] \;=\; \sum_{x \in \Omega} g(x)\,\mathrm{P}(x)$$

  i.e., an expected value is a kind of weighted average
- If we're given an alignment $\boldsymbol{a}$, we can calculate translation counts $n_{f,e}(\boldsymbol{a})$ directly
- If we're given a probability distribution $\mathrm{P}(\boldsymbol{a})$ over alignments, we can calculate the *expected translation counts*:

$$\mathrm{E}[n_{f,e}] \;=\; \sum_{\boldsymbol{a}} n_{f,e}(\boldsymbol{a})\,\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$$

A single iteration of the EM algorithm

- At the beginning of iteration t, the current estimate of the word translation probabilities is $\boldsymbol{\tau}^{(t)}$:
  1. *E-step:*
     a. For each possible alignment $\boldsymbol{a}$, calculate $\mathrm{P}_{\boldsymbol{\tau}^{(t)}}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$
     b. Summing over all possible alignments $\boldsymbol{a}$, calculate expected translation counts

$$\mathrm{E}_{\boldsymbol{\tau}^{(t)}}[n_{f,e}] \quad = \quad \sum_{\boldsymbol{a}} n_{f,e}(\boldsymbol{a}) \, \mathrm{P}_{\boldsymbol{\tau}^{(t)}}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$$

  2. *M-step:* Use the expected counts to update $\boldsymbol{\tau}$:

$$\tau_{f,e}^{(t+1)} \quad = \quad \frac{\mathrm{E}_{\boldsymbol{\tau}^{(t)}}[n_{f,e}]}{\sum_{f} \mathrm{E}_{\boldsymbol{\tau}^{(t)}}[n_{f,e}]}$$

- Unfortunately, the algorithm as described is infeasible ...

Dynamic programming for expected counts

- By definition, every French word f_k in the corpus is aligned with some English word e_{a_k}, but *which one*?
- Because we assume $\mathrm{P}_{\boldsymbol{\tau}}(\boldsymbol{a} \mid \boldsymbol{e})$ is uniform, alignment probability only depends on $\boldsymbol{\tau}$. With some algebra, can show:

$$\mathrm{P}_{\boldsymbol{\tau}}(a_k = j \mid \boldsymbol{e}, f_k) \;=\; \frac{\tau_{f_k, e_j}}{\sum_{j'=0}^{|\boldsymbol{e}|} \tau_{f_k, e_{j'}}}$$

- Expected translation counts:

$$\mathrm{E}_{\boldsymbol{\tau}}[n_{f', e'}] \;=\; \sum_{\substack{k : f_k = f' \\ j : e_j = e'}} \mathrm{P}_{\boldsymbol{\tau}}(a_k = j \mid \boldsymbol{e}, f_k)$$

- Informally, to compute transition counts iterate through each French word f_k in parallel corpus
  - compute probability $\mathrm{P}_{\boldsymbol{\tau}}(a_k = j \mid \boldsymbol{e}, f_k)$ that it is aligned to e_j
  - add $\mathrm{P}_{\boldsymbol{\tau}}(a_k = j \mid \boldsymbol{e}, f_k)$ to expected counts $\mathrm{E}_{\boldsymbol{\tau}}[n_{f_k, e_j}]$

MACQUARIE UNIVERSITY

EM algorithm for learning alignments and translation probabilities

```
Initialise translation probabilities $\boldsymbol{\tau}$ somehow (e.g., randomly)
Repeat for $t = 0, 1, \ldots$ until convergence:
    Initialise expected counts $\mathrm{E}[n_{f,e}] = 0$ for all $e, f$
    For each French word position $k \in 1, \ldots, |\boldsymbol{f}|$:
        For each English word position $j \in 1, \ldots, |\boldsymbol{e}|$:
            $\mathrm{E}[n_{f_k,e_j}] \mathrel{+}= \mathrm{P}_{\boldsymbol{\tau}}(a_k = j \mid \boldsymbol{e}, f_k) = \frac{\tau_{f_k,e_j}}{\sum_{j'=0}^{|\boldsymbol{e}|} \tau_{f_k,e_{j'}}}$
    Recalulate translation probabilities $\boldsymbol{\tau}$:
        $\tau_{f,e} = \frac{\mathrm{E}[n_{f,e}]}{\mathrm{E}[n_{\cdot,e}]}$
```

EM algorithm in pictures: $\mathrm{P}(\boldsymbol{f} \mid \boldsymbol{e})$

	◊	They	have	full	access	to	working	documents
Ils	0.01	0.2	0.001	0.001	0.001	0.001	0.001	0.001
ont	0.01	0.001	0.2	0.001	0.001	0.001	0.001	0.001
accès	0.01	0.001	0.001	0.001	0.2	0.001	0.001	0.001
à	0.01	0.001	0.001	0.001	0.001	0.2	0.001	0.001
tous	0.01	0.001	0.001	0.1	0.001	0.001	0.001	0.001
le	0.01	0.001	0.001	0.001	0.001	0.001	0.001	0.001
documents	0.01	0.001	0.001	0.001	0.001	0.001	0.001	0.2
de	0.01	0.001	0.001	0.001	0.001	0.001	0.001	0.001
travail	0.01	0.001	0.001	0.001	0.001	0.001	0.1	0.001

MACQUARIE UNIVERSITY

EM algorithm in pictures: $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e}, \boldsymbol{f})$

	$\lozenge$	They	have	full	access	to	working	documents
Ils	0.05	0.9	0.001	0.001	0.001	0.001	0.001	0.001
ont	0.05	0.001	0.9	0.001	0.001	0.001	0.001	0.001
accès	0.05	0.001	0.001	0.001	0.9	0.001	0.001	0.001
à	0.05	0.001	0.001	0.001	0.001	0.9	0.001	0.001
tous	0.1	0.001	0.001	0.5	0.001	0.001	0.001	0.001
le	0.8	0.001	0.001	0.001	0.001	0.001	0.001	0.001
documents	0.05	0.001	0.001	0.001	0.001	0.001	0.001	0.9
de	0.8	0.001	0.001	0.001	0.001	0.001	0.001	0.001
travail	0.1	0.001	0.001	0.001	0.001	0.001	0.5	0.001

MACQUARIE UNIVERSITY

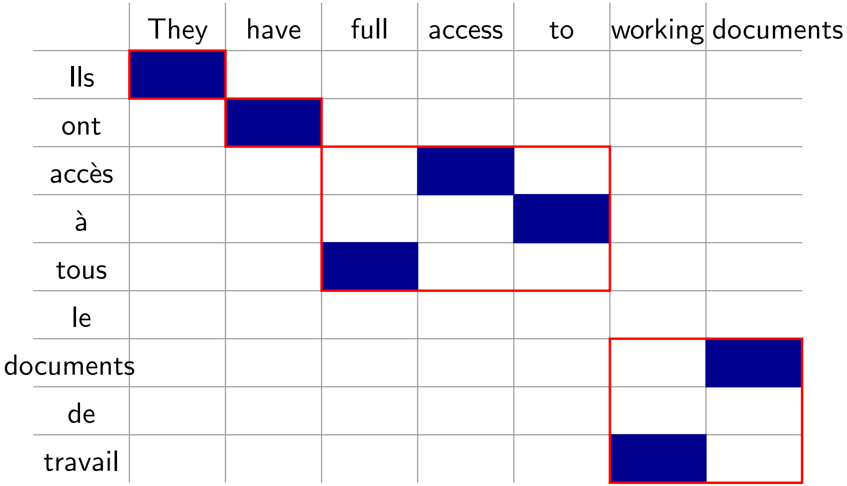
Alignments learnt from Canadian Hansards

count	English	French	count	English	French
306043	,	,	289093	the	le
246182	.	.	243395	◊	.
213067	of	de	192687	and	et
157561	the	de	106313	the	la
100649	to	à	99661.1	◊	le
96360.3	to	de	87601.5	.	de
84200.4	that	que	84127	we	nous
83155.8	is	est	81717.6	◊	de
75641.3	the	les	73189.4	:	:
65337.2	a	un	63712.4	not	pas
62694.6	not	ne	57248.2	I	je
56526.6	government	gouvernement	54257.5	i	je
51031.5	-	-	47926.5	a	une
47682.4	the	à	46082.2	in	dans
45835.1	Mr.	.	44350.8	of	le
44127.6	.	le	42532.5	Speaker	Président
41468.1	in	de	38837.2	to	le
36390.9	,	de	34435.7	?	?
3441[illegible]	,	le	33032.2	for	de

From IBM model 1 to a machine translation system

- IBM model 1 produces surprisingly accurate alignments and word translation probabilities
- For translation you need a nontrivial *alignment model* $\mathrm{P}(\boldsymbol{a} \mid \boldsymbol{e})$
  - IBM model 2 learns a probabilistic mapping of French word positions to English word positions
  - More sophisticated models are based on *phrasal reorderings*
- Increasing interest in using syntactic parsing to define phrasal translation units and reordering models
- The problem of finding the optimal translation is usually NP-hard
  - beam search for optimal translation

Identifying phrases from word alignments



	They	have	full	access	to	working	documents
Ils	■						
ont		■					
accès				■			
à					■		
tous			■				
le							
documents							■
de							
travail						■	

Outline

MACQUARIE UNIVERSITY

Sequence tagging overview

- Sequence tagging:
  given a sequence $(x_1, \ldots, x_n)$ of *observations* (where each $x_i \in \mathcal{X}$),
  return a sequence $(y_1, \ldots, y_n)$ of *states* or *labels* (where each $y_i \in \mathcal{Y}$)
- Hidden Markov Models, and their three different representations:
  - stochastic automata
  - Bayes nets
  - Trellis
- Even though there are *exponentially many label sequences*, there are *linear time dynamic programming algorithms*
  - *Viterbi algorithm* for finding most likely label sequence
  - *Forward-backward algorithm* for finding expected counts
- Expectation Maximisation algorithm for learning HMMs from strings alone

Applications of sequence tagging

- *Part of speech (POS) tagging:* $\boldsymbol{x}$ are the words of a sentence, $\boldsymbol{y}$ are their parts of speech.

$\boldsymbol{y}$:	DT	JJ	NN	VBD	NNP	.
$\boldsymbol{x}$:	the	big	cat	bit	Sam	.

- *Noun-phrase chunking:* $\boldsymbol{x}$ are the words of a sentence, $\boldsymbol{y}$ indicates whether they are in the beginning, middle or end of a noun phrase (NP) chunk.

$\boldsymbol{y}$:	[NP	NP	NP]	_	[NP]	.
$\boldsymbol{x}$:	the	big	cat	bit	Sam	.

- *Named entity detection:* The $\boldsymbol{x}$ are the words of a sentence, $\boldsymbol{y}$ indicates whether they are in the beginning, middle or end of a noun phrase (NP) chunk that is the name of a person, company or location.

$\boldsymbol{y}$:	[CO	CO]	_	[LOC]	_	[PER]	_
$\boldsymbol{x}$:	XYZ	Corp.	of	Boston	announced	Spade's	resignation

- *Speech recognition:* $\boldsymbol{x}$ are 100 msec. time slices of acoustic input, and $\boldsymbol{y}$ are the corresponding phonemes (i.e., y_i is the phoneme being uttered in time slice x_i)

Hidden Markov Models (HMMs)

- A *Markov model* generates sequence $\boldsymbol{y} = (y_0 = \triangleright, y_1, \ldots, y_n, y_{n+1} = \triangleleft)$ by generating each y_i from y_{i-1}

$$\mathrm{P}(\boldsymbol{y}) \;=\; \prod_{i=1}^{n+1} \mathrm{P}(y_i \mid y_{i-1}) \;=\; \prod_{i=1}^{n+1} \sigma_{y_i, y_{i-1}}$$

- In a *Hidden Markov Model:*
  - the *state sequence* $\boldsymbol{y} = (y_1, \ldots, y_n)$ is generated by a Markov model
  - each x_i in the *observation sequence* $\boldsymbol{x} = (x_1, \ldots, x_n)$ is generated from its corresponding state y_i

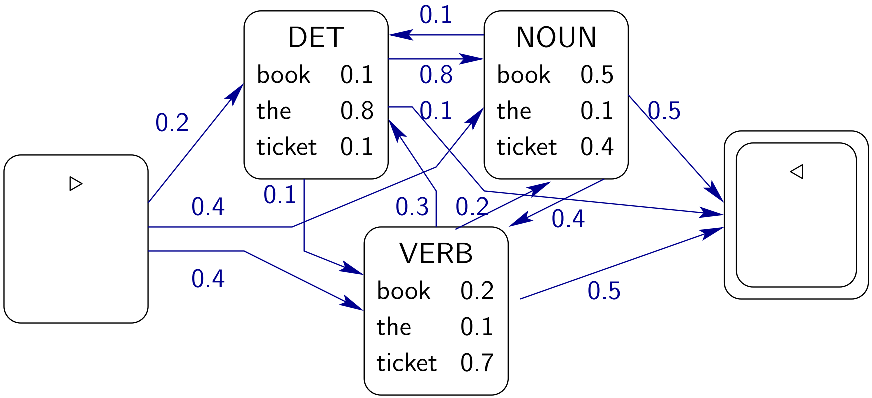
$$\mathrm{P}(\boldsymbol{x} \mid \boldsymbol{y}) \;=\; \prod_{i=1}^{n} \mathrm{P}(x_i \mid y_i) \;=\; \prod_{i=1}^{n} \tau_{x_i, y_i}$$

  - so the joint distribution over $(\boldsymbol{x}, \boldsymbol{y})$ is:

$$\mathrm{P}(\boldsymbol{x}, \boldsymbol{y}) \;=\; \left(\prod_{i=1}^{n} \mathrm{P}(y_i \mid y_{i-1})\, \mathrm{P}(x_i \mid y_i)\right) \mathrm{P}(\triangleleft \mid y_n) \;=\; \left(\prod_{i=1}^{n} \sigma_{y_i, y_{i-1}} \tau_{x_i, y_i}\right) \sigma_{\triangleleft, y_n}$$

MACQUARIE UNIVERSITY

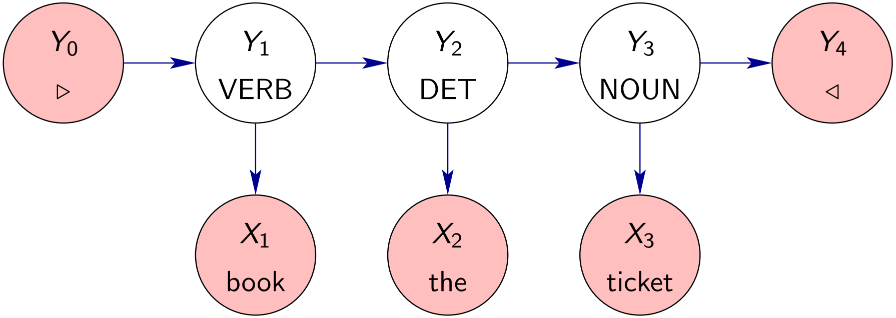
HMMs as stochastic automata (Moore machines)



- Automaton depicts all possible $(\boldsymbol{x}, \boldsymbol{y})$ sequences
- States are nodes in graph (identified with a label)
- State-to-state transition probabilities $\boldsymbol{\sigma}$ are arc labels
- State-to-observation emission probabilities $\boldsymbol{\tau}$ are nodel labels

MACQUARIE UNIVERSITY

HMMs as Bayes nets



- Each random variable (e.g., observation x_i and label y_i) is represented by a node
- Shows factorisation of joint distribution $\mathrm{P}(\boldsymbol{x}, \boldsymbol{y})$ into product of conditional distributions
- Each node is associated with a conditional distribution from each of its parents

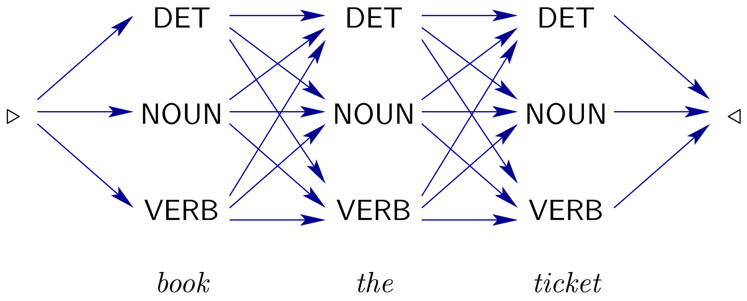
Important problems for HMMs

1. What's the probability of a sequence of *states and observations* $\mathrm{P}(\boldsymbol{x}, \boldsymbol{y})$?
2. What's the probability of a sequence of *observations* $\mathrm{P}(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Y}^n} \mathrm{P}(\boldsymbol{x}, \boldsymbol{y})$?
3. Given a sequence of observations $\boldsymbol{x}$, what is the *most likely sequence of states* $\mathrm{argmax}_{\boldsymbol{y} \in \mathcal{Y}^n} \mathrm{P}(\boldsymbol{y} \mid \boldsymbol{x})$?
4. Given *visible training data* $(\boldsymbol{x}, \boldsymbol{y})$, estimate the HMM parameters $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$
5. Given *hidden training data* $\boldsymbol{x}$, estimate the HMM parameters $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$

- Problems 2, 3 and 5 involve summing over all $\boldsymbol{y} \in \mathcal{Y}^n$,
- but this can be done in *linear time* using *dynamic programming* over a *trellis*

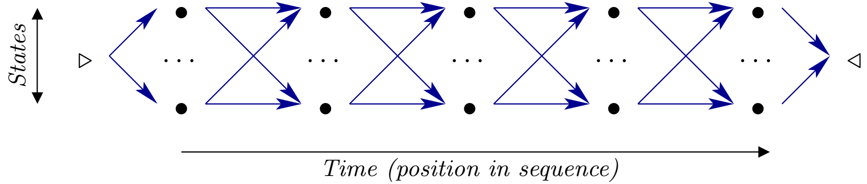
MACQUARIE UNIVERSITY

HMM computation using a trellis



- The *trellis* represents *all possible state sequences $\boldsymbol{y} \in \mathcal{Y}^n$ for a fixed observation sequence $\boldsymbol{x} = (x_1, \ldots, x_n)$*
- One node for each (y, i) combination for $y \in \mathcal{Y}$, $i \in 1, \ldots, n$
- The *weight of an edge* from $(y', i-1)$ to (y, i) is $\sigma_{y,y'}\tau_{x_i,y}$, i.e., *probability of moving from y' to y and emitting x_i*
- The *weight of a path* is the *product* of the weights of the edges in it
- Each state sequence $\boldsymbol{y}$ identifies a path from ▷ to ◁ in trellis
- $\mathrm{P}(\boldsymbol{x}, \boldsymbol{y})$ = *weight of path $\boldsymbol{y}$*

Calculating the probability of an observation sequence $\boldsymbol{x}$



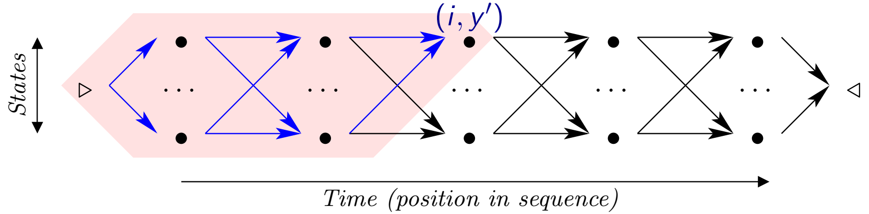
- Goal: compute probability $\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x})$ of observed sequence $\boldsymbol{x}$

$$\begin{aligned}\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x}) &= \sum_{\boldsymbol{y}\in\mathcal{Y}^n} \mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x},\boldsymbol{y}) \\ &= \sum_{\boldsymbol{y}\in\mathcal{Y}^n} \left(\prod_{i=1}^{n} \sigma_{y_i,y_{i-1}} \tau_{x_i,y_i}\right) \sigma_{\triangleleft,y_n}\end{aligned}$$

- $\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x})$ is the *sum of the probability of all paths* in trellis
- The number of sequences in $\mathcal{Y}^n$ grows exponentially with n

$\Rightarrow$ *calculating $\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x})$ by direct summation is infeasible*

Forward probabilities $\boldsymbol{\alpha}$



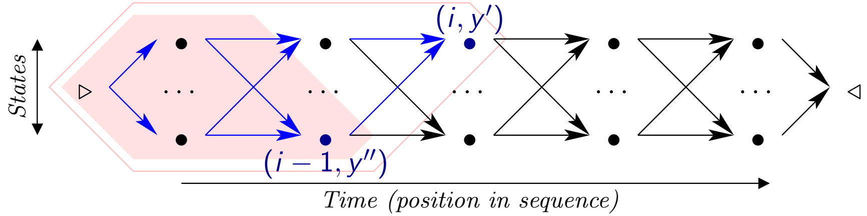
- *Forward probability* $\alpha_{i,y'}$ is the probability of generating $(x_1, \ldots, x_i)$ ending in a state y'

$$\alpha_{i,y'} \quad = \quad \sum_{\mathbf{y} \in \mathcal{Y}^{i-1}} \mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{Y}_{1:i-1} = \boldsymbol{y}, Y_i = y', Vx_{1:i})$$

- $\alpha_{i,y'}$ is sum of weights of all paths from $\triangleleft$ to (i, y')
- $\boldsymbol{\alpha}$ is an array indexed by *a position* i and a state $y' \in \mathcal{Y}$
- Probability of observations $\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x}) = \alpha_{n+1,\triangleleft}$

MACQUARIE UNIVERSITY

Dynamic programming for forward probabilities α



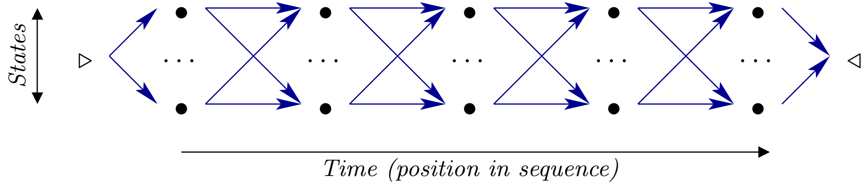
- With simple algebra it's possible to show that:

$$\alpha_{i,y'} \quad = \quad \sum_{y'' \in \mathcal{Y}} \alpha_{i-1,y''} \left(\sigma_{y',y''} \, \tau_{x_i,y'}\right)$$

$\Rightarrow$ *Dynamic programming algorithm for calculating α*

```
Set $\alpha_{0,\triangleleft} = 1$ and $\alpha_{0,y} = 0$ for all other $y \in \mathcal{Y}$
For $i = 1, \ldots, n$:
    For $y' \in \mathcal{Y}$:
        $\alpha_{i,y'} = \sum_{y'' \in \mathcal{Y}} \alpha_{i-1,y''} (\sigma_{y',y''} \, \tau_{x_i,y'})$
```

Finding the most likely state sequence

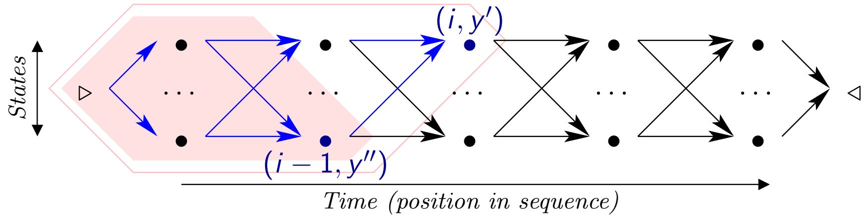


- Goal: find the most likely state sequence $\boldsymbol{Y}^\star(\boldsymbol{x})$ wrt HMM $(\boldsymbol{\sigma}, \boldsymbol{\tau})$

$$\boldsymbol{Y}^\star(\boldsymbol{x}) \;=\; \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{Y}^n} \mathrm{P}(\boldsymbol{y} \mid \boldsymbol{x}) \;=\; \operatorname*{argmax}_{\boldsymbol{y}} \mathrm{P}(\boldsymbol{y}, \boldsymbol{x})$$

  - This is how you use an HMM to tag an observation sequence $\boldsymbol{x}$
- Key observation: *best path from $\triangleright$ to (i, y') must begin with the best path to $(i-1, y'')$ for some $y'' \in \mathcal{Y}$*

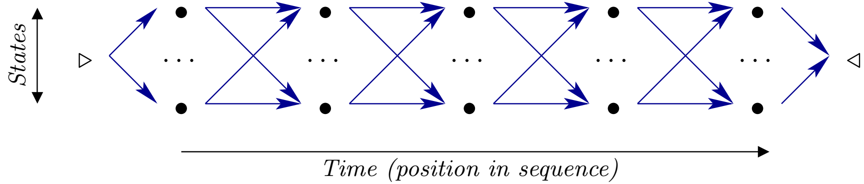
The Viterbi algorithm for most likely state sequence



- Compute best path to every node (i, y') as well as its weight
- Algorithm overview:

```
For $i = 1, \ldots, n+1$:
  for each $y' \in \mathcal{Y}$:
    calculate best path to $(i, y')$ and its probability
    using best paths to $(i-1, y'')$ and their probabilities
```

The Viterbi algorithm for HMMs and shortest path algorithms



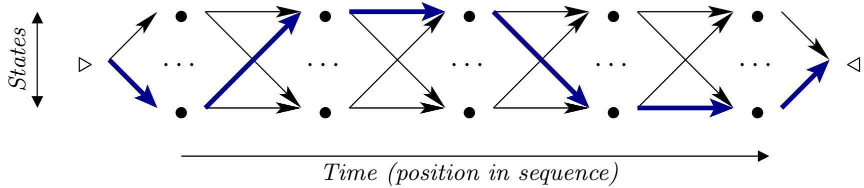
- If we define the *weight* of an edge from $(y', i-1)$ to (y, i) as its *negative log probability*

$$w_{(y',i-1),(y,i)} = -\log \sigma_{y,y'} \tau_{x_i,y}$$

- Then the *negative log probability of path is the sum of its weights*

$\Rightarrow$ The highest probability state sequence is the *shortest path from ▷ to ◁*

Estimating HMMs from *visible* training data



- Training data $D = (\boldsymbol{x}, \boldsymbol{y}) = ((x_1, \ldots, x_n), (y_1, \ldots, y_n))$, i.e., observations and their states
  - E.g., *They/PRP liked/VBD the/DT Old/NNP Man/NNP*
  - Identifies a *unique path in the trellis*
- Because HMMs are products of multinomials, the MLEs are *relative frequencies*

$$\widehat{\sigma}_{y,y'} = \frac{n_{y,y'}(D)}{n_{\cdot,y'}(D)}$$

$$\widehat{\tau}_{x,y} = \frac{n_{x,y}(D)}{n_{\cdot,y}(D)}$$

Estimating HMMs from observations alone

- Training data $D = \boldsymbol{x} = (x_1, \ldots, x_n)$, i.e., observations but not the states
  - E.g., *They liked the Old Man*
  - State sequence could be any path in the trellis
- *Expectation Maximization algorithm* for estimating HMMs from observations alone:

```
Initialise $\boldsymbol{\sigma}^{(0)}, \boldsymbol{\tau}^{(0)}$ somehow (e.g., randomly)
For iteration $t = 0, 1, 2, \ldots$ until converged:
    E-step: Calculate $\mathrm{E}_{\boldsymbol{\sigma}^{(t)},\boldsymbol{\tau}^{(t)}}[n_{y,y'}]$ and $\mathrm{E}_{\boldsymbol{\sigma}^{(t)},\boldsymbol{\tau}^{(t)}}[n_{x,y}]$
    M-step: Calculate $\boldsymbol{\sigma}^{(t+1)}$ and $\boldsymbol{\tau}^{(t+1)}$
```

$$\sigma^{(t+1)}_{y,y'} = \frac{\mathrm{E}_{\boldsymbol{\sigma}^{(t)},\boldsymbol{\tau}^{(t)}}[n_{y,y'}]}{\mathrm{E}_{\boldsymbol{\sigma}^{(t)},\boldsymbol{\tau}^{(t)}}[n_{\cdot,y'}]}$$

$$\tau^{(t+1)}_{x,y} = \frac{\mathrm{E}_{\boldsymbol{\sigma}^{(t)},\boldsymbol{\tau}^{(t)}}[n_{x,y}]}{\mathrm{E}_{\boldsymbol{\sigma}^{(t)},\boldsymbol{\tau}^{(t)}}[n_{\cdot,y}]}$$

Calculating expected counts by enumeration

- Training data $D = \boldsymbol{x} = (x_1, \ldots, x_n)$ (no state information)
- Given model $\boldsymbol{\sigma}, \boldsymbol{\tau}$, the expected values are as follows:

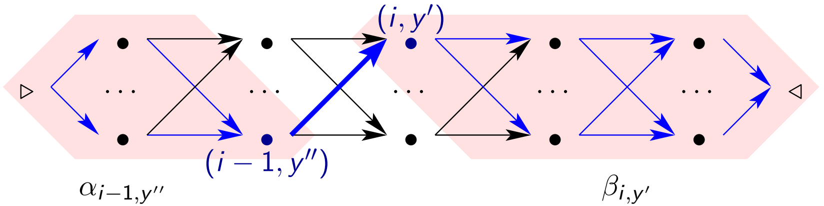
$$\begin{aligned}
\mathrm{E}[n_{y',y''}] &= \sum_{\boldsymbol{y} \in \mathcal{Y}^n} n_{y',y''}(\boldsymbol{x}, \boldsymbol{y}) \, \mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{y} \mid \boldsymbol{x}) \\
\mathrm{E}[n_{x',y'}] &= \sum_{\boldsymbol{y} \in \mathcal{Y}^n} n_{x',y'}(\boldsymbol{x}, \boldsymbol{y}) \, \mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{y} \mid \boldsymbol{x}), \text{ where:} \\
\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{y} \mid \boldsymbol{x}) &= \frac{\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x}, \boldsymbol{y})}{\sum_{\boldsymbol{y}' \in \mathcal{Y}^n} \mathrm{P}(\boldsymbol{x}, \boldsymbol{y})} \\
\mathrm{P}_{\boldsymbol{\sigma},\boldsymbol{\tau}}(\boldsymbol{x}, \boldsymbol{y}) &= \left(\prod_{i=1}^{n} \sigma_{y_i,y_{i-1}} \tau_{x_i,y_i} \right) \sigma_{\triangleleft,y_n}
\end{aligned}$$

- The *number of state sequences $\boldsymbol{y}$ grows exponentially with n*

$\Rightarrow$ Enumerating all state sequences is infeasible except when $\boldsymbol{x}$ is very short.

MACQUARIE UNIVERSITY

Dynamic programming for computing expected counts



- High level description:
  - compute *expected number of times each edge in trellis is traversed* $\mathrm{E}[N_{(i,y'),(i-1,y'')}]$
  - sum these over positions $i \in 1, \ldots, n+1$ to compute $\mathrm{E}[N_{y',y''}]$
- $\mathrm{E}[N_{(i,y'),(i-1,y'')}]$ is computed using *forward probability* $\alpha_{i-1,y''}$ and *backward probability* $\beta_{i,y'}$
- *Backward probability* $\beta_{i,y'}$ is sum of weights of all paths from (i, y') to $\triangleleft$
  - there is a dynamic programming algorithm for $\beta_{i,y'}$

Learning a 17-state HMM with EM from Penn WSJ corpus

count	**words generated by state**
123911	of in , and for to from on at with as by ’s or ; than that – rose after
116885	the a an its this his new their one last that any chief another her no our s
116262	company year market share quarter week group time u.s. month board sh
102328	sales trading prices president companies one it more stocks bonds investor
76547.5	to n’t be been not have more up “ out expected and it also so going do b
76250	’s new stock first “ york u.s. federal big and market major national last fi
66005.4	is to are was will has would were have had could and can may ’s do did d
65616.9	in to by for on with at that as from about than up into be have make und
65097.6	it mr. he they which we i there who you that she this analysts what howe
58808.6	other their its some u.s. these “ many more and as such all net those new
57329.6	, that and – ” who because : also ; “ by as such but -lrb- when ago out y
53400.4	. ” ? : -rrb- ago ... earlier outstanding – ! ’ yesterday “ in here today of
46854.1	“ ” but and said that as says if when which in while even some including
43697.8	, ’s said and is says was say has had inc earlier added think co reported ar
40517.8	$ to or about 10 two 8 1 up oct. three 50 20 15 2 five at nov. down 30
33486.3	corp. inc. & co. also -rrb- officials , bush group ’s -lrb- ltd. d. internation
30768	% million billion years cents months -rrb- days shares yen points 1 30 ago

Another EM run on same corpus

count	**words generated by state**
118833	the a $ an its this his their one last that any chief another her no each ou
116130	million year company market share billion quarter shares president week ti
105943	of in and for to on from at with 's as by or ; after over ' is under about
100047	% years cents sales trading prices it companies stocks bonds them days ea
84530.4	, that and – also : because -lrb- ; " who as said " than when but -rrb- ou
69977.6	$ million a % new first " billion u.s. stock big major last next own most t
69784.6	to be n't been not have more " out expected it up also so do going well s
67787.4	in to by for at on with as from about than that up into make sell buy thro
62922.6	it he they " which we i there who that you she analysts however this and
62719.6	other their its two some these u.s. new three net many 8 such all more 10
55219.1	is are was will has would were have had 's could to can may do did does s
52359.5	. " ? : -rrb- – ... ! ' " in ends of , activity -rcb- shares daffynition off mil
52252.1	's and corp. inc. & co. york stock exchange group officials street bank sec
47304.5	" mr. but " and that if when as some which while many even president be
40179.2	to and " more up or ago according earlier such yesterday compared based
36356.3	said says and is 's or say was has including " had added reported are -rrb-
31420.7	new american company federal san wall national u.s. dow general first we

Further topics on HMMs and sequence modelling

- Wide range of applications for sequence modelling in computational linguistics
- Higher-order HMMs
  - The HMMs we investigated here are *first-order*, i.e., the next state y_i is generated from y_{i-1}
  - In a k-th order HMM, y_i is generated from $y_{i-1}, \ldots, y_{i-k}$
- This theory generalises to *stochastic finite-state transducers* (useful for modelling morphology)
- It is possible to train HMMs *discriminatively* to maximise $\mathrm{P}(\boldsymbol{y} \mid \boldsymbol{x})$; such models are called *Conditional Random Fields*

Outline

MACQUARIE UNIVERSITY

Why grammars?

Grammars specify a wide range of *sets of structured objects*

- especially useful for describing human languages
- applications in vision, computational biology, etc

There is a *hierarchy* of kinds of grammars

- if a language can be specified by a grammar low in the hierarchy, then it can be specified by a grammar higher in the hierarchy
- the location of a grammar in this hierarchy determines its computational properties

There are generic algorithms for *computing with* and *estimating* (learning) each kind of grammar

- no need to devise new models and corresponding algorithms for each new set of structures

Preview of (P)CFG material

- Why are context-free grammars called "context-free"?
- Context-free grammars (CFG) derivations and parse trees
- Probabilistic CFGs (PCFGs) define probability distributions over derivations/trees
- The number of derivations often grows exponentially with sentence length
- Even so, we can compute the sum/max of probabilities of all trees in cubic time
- It's easy to estimate PCFGs from a treebank (a sequence of trees)
- The EM algorithm can estimate PCFGs from a corpus of strings

Formal languages

$\mathcal{W}$ is a finite set of *terminal symbols*, the vocabulary of the language

- E.g., $\mathcal{W} = \{\text{likes}, \text{Sam}, \text{Sasha}, \text{thinks}\}$

A *string* is a *finite sequence* of elements of $\mathcal{W}$

- E.g., $\text{Sam thinks Sam likes Sasha}$

$\mathcal{W}^\star$ is the set of all strings (including the *empty string* ϵ)
$\mathcal{W}^+$ is the set of all non-empty strings
A (formal) *language* is a set of strings (a subset of $\mathcal{W}^\star$)

- E.g., $L = \{\text{Sam}, \text{Sam thinks}, \text{Sasha thinks}, \ldots\}$

A *probabilistic language* is a probability distribution over a language

Rewrite grammars

A rewrite grammar $G = (\mathcal{W}, \mathcal{N}, S, \mathcal{R})$ consists of

- $\mathcal{W}$, a finite set of *terminal symbols*
- $\mathcal{N}$, a finite set of *nonterminal symbols* disjoint from $\mathcal{W}$
- $S \in \mathcal{N}$ is the *start symbol*, and
- $\mathcal{R}$ is a finite subset of $\mathcal{N}^+ \times (\mathcal{N} \cup \mathcal{W})^\star$

The members of $\mathcal{R}$ are called *rules* or *productions*, and usually written $\alpha \to \beta$, where $\alpha \in \mathcal{N}^+$ and $\beta \in (\mathcal{N} \cup \mathcal{W})^\star$

A rewrite grammar defines the *rewrites relation* $\Rightarrow$, where $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ iff $\alpha \to \beta \in \mathcal{R}$ and $\gamma, \delta \in (\mathcal{N} \cup \mathcal{W})^\star$.

A *derivation* of a string $\boldsymbol{w} \in \mathcal{W}^\star$ is a finite sequence of rewritings $S \Rightarrow \ldots \Rightarrow \boldsymbol{w}$.

$\Rightarrow^\star$ is the reflexive transitive closure of $\Rightarrow$

The language *generated* by G is $\{\boldsymbol{w} : S \Rightarrow^\star \boldsymbol{w}, \text{where } \boldsymbol{w} \in \mathcal{W}^\star\}$

Example of a rewriting grammar

$G_1 = (\mathcal{W}_1, \mathcal{N}_1, \mathrm{S}, \mathcal{R}_1)$, where

$$\mathcal{W}_1 = \{\mathrm{Al}, \mathrm{George}, \mathrm{snores}\},$$
$$\mathcal{N}_1 = \{\mathrm{S}, \mathrm{NP}, \mathrm{VP}\},$$
$$\mathcal{R}_1 = \{\mathrm{S} \to \mathrm{NP\ VP}, \mathrm{NP} \to \mathrm{Al}, \mathrm{NP} \to \mathrm{George}, \mathrm{VP} \to \mathrm{snores}\}.$$

Sample derivations:

$$\mathrm{S} \Rightarrow \mathrm{NP\ VP} \Rightarrow \mathrm{Al\ VP} \Rightarrow \mathrm{Al\ snores}$$
$$\mathrm{S} \Rightarrow \mathrm{NP\ VP} \Rightarrow \mathrm{George\ VP} \Rightarrow \mathrm{George\ snores}$$

MACQUARIE UNIVERSITY

The Chomsky Hierarchy

Grammars classified by the shape of their productions $\alpha \to \beta$.

Context-sensitive: $|\alpha| \leq |\beta|$

Context-free: $|\alpha| = 1$

Right-linear: $|\alpha| = 1$ and $\beta \in \mathcal{W}^\star\,(\mathcal{N} \cup \epsilon)$.

The classes of languages generated by these classes of grammars form a strict hierarchy (ignoring ϵ).

Language class	*Recognition complexity*
Unrestricted	undecidable
Context-sensitive	exponential time
Context-free	polynomial time
Linear	linear time

Right linear grammars define *finite state languages*, and probabilistic right linear grammars define the same distributions as Hidden Markov Models.

Context-sensitivity in human languages

- Some human languages are not context-free (Shieber 1984, Culy 1984).
- Context-sensitive grammars don't seem useful for describing human languages.
- Trees are intuitive descriptions of linguistic structure and are *normal forms* for context-free grammar derivations.
- There is an infinite hierarchy of language families (and grammars) between context-free and context-sensitive.
- *Mildly context-sensitive grammars*, such as Tree Adjoining Grammars (Joshi) and Combinatory Categorial Grammar (Steedman) seem useful for natural languages.

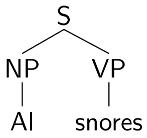
Parse trees for context-free grammars

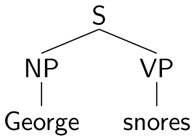
A *parse tree* generated by CFG $G = (\mathcal{W}, \mathcal{N}, S, \mathcal{R})$ is a finite ordered tree labeled with labels from $\mathcal{N} \cup \mathcal{W}$, where:

- the root node is labeled S
- for each node n labeled with a nonterminal $A \in \mathcal{N}$ there is a rule $A \to \beta \in \mathcal{R}$ and n's children are labeled β
- each node labeled with a terminal has no children

$\mathcal{T}_G$ is the set of all parse trees generated by G.
$\mathcal{T}_G(\boldsymbol{w})$ is the subset of $\mathcal{T}_G$ with yield $\boldsymbol{w} \in \mathcal{W}^\star$.

$$\mathcal{R} = \{\mathrm{S} \to \mathrm{NP\ VP}, \mathrm{NP} \to \mathrm{Al}, \mathrm{NP} \to \mathrm{George}, \mathrm{VP} \to \mathrm{snores}\}$$





Example of a CF derivation and parse tree

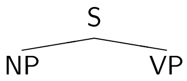
$$\mathcal{R} = \left\{ \begin{array}{ccc} \mathrm{S} \to \mathrm{NP\ VP} & \mathrm{NP} \to \mathrm{D\ N} & \mathrm{VP} \to \mathrm{V} \\ \mathrm{D} \to \mathrm{the} & \mathrm{N} \to \mathrm{dog} & \mathrm{V} \to \mathrm{barks} \end{array} \right\}$$

S

S

Example of a CF derivation and parse tree

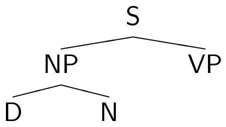
$$\mathcal{R} = \left\{ \begin{array}{ccc} \text{S} \to \text{NP VP} & \text{NP} \to \text{D N} & \text{VP} \to \text{V} \\ \text{D} \to \text{the} & \text{N} \to \text{dog} & \text{V} \to \text{barks} \end{array} \right\}$$



$$\begin{array}{c} \text{S} \\ \Rightarrow \quad \text{NP VP} \end{array}$$

Example of a CF derivation and parse tree

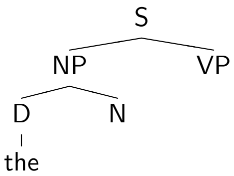
$$\mathcal{R} = \left\{ \begin{array}{ccc} \mathrm{S} \to \mathrm{NP\ VP} & \mathrm{NP} \to \mathrm{D\ N} & \mathrm{VP} \to \mathrm{V} \\ \mathrm{D} \to \mathrm{the} & \mathrm{N} \to \mathrm{dog} & \mathrm{V} \to \mathrm{barks} \end{array} \right\}$$



$$\begin{array}{cc} & \mathrm{S} \\ \Rightarrow & \mathrm{NP\ VP} \\ \Rightarrow & \mathrm{D\ N\ VP} \end{array}$$

Example of a CF derivation and parse tree

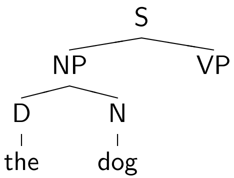
$$\mathcal{R} = \left\{ \begin{array}{ccc} \mathrm{S} \to \mathrm{NP\ VP} & \mathrm{NP} \to \mathrm{D\ N} & \mathrm{VP} \to \mathrm{V} \\ \mathrm{D} \to \mathrm{the} & \mathrm{N} \to \mathrm{dog} & \mathrm{V} \to \mathrm{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \mathrm{S} \\ \Rightarrow & \mathrm{NP\ VP} \\ \Rightarrow & \mathrm{D\ N\ VP} \\ \Rightarrow & \mathrm{the\ N\ VP} \end{array}$$

Example of a CF derivation and parse tree

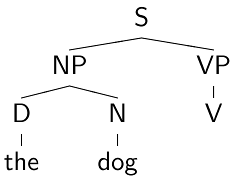
$$\mathcal{R} = \left\{ \begin{array}{ccc} \text{S} \to \text{NP VP} & \text{NP} \to \text{D N} & \text{VP} \to \text{V} \\ \text{D} \to \text{the} & \text{N} \to \text{dog} & \text{V} \to \text{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \text{S} \\ \Rightarrow & \text{NP VP} \\ \Rightarrow & \text{D N VP} \\ \Rightarrow & \text{the N VP} \\ \Rightarrow & \text{the dog VP} \end{array}$$

Example of a CF derivation and parse tree

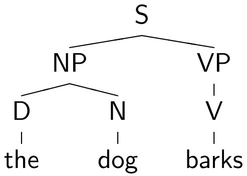
$$\mathcal{R} = \left\{ \begin{array}{ccc} \text{S} \to \text{NP VP} & \text{NP} \to \text{D N} & \text{VP} \to \text{V} \\ \text{D} \to \text{the} & \text{N} \to \text{dog} & \text{V} \to \text{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \text{S} \\ \Rightarrow & \text{NP VP} \\ \Rightarrow & \text{D N VP} \\ \Rightarrow & \text{the N VP} \\ \Rightarrow & \text{the dog VP} \\ \Rightarrow & \text{the dog V} \end{array}$$

Example of a CF derivation and parse tree

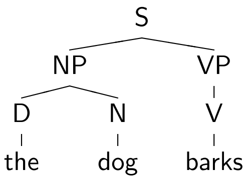
$$\mathcal{R} = \left\{ \begin{array}{ccc} \mathrm{S} \to \mathrm{NP\ VP} & \mathrm{NP} \to \mathrm{D\ N} & \mathrm{VP} \to \mathrm{V} \\ \mathrm{D} \to \mathrm{the} & \mathrm{N} \to \mathrm{dog} & \mathrm{V} \to \mathrm{barks} \end{array} \right\}$$



$$\begin{array}{rc} & \mathrm{S} \\ \Rightarrow & \mathrm{NP\ VP} \\ \Rightarrow & \mathrm{D\ N\ VP} \\ \Rightarrow & \mathrm{the\ N\ VP} \\ \Rightarrow & \mathrm{the\ dog\ VP} \\ \Rightarrow & \mathrm{the\ dog\ V} \\ \Rightarrow & \mathrm{the\ dog\ barks} \end{array}$$

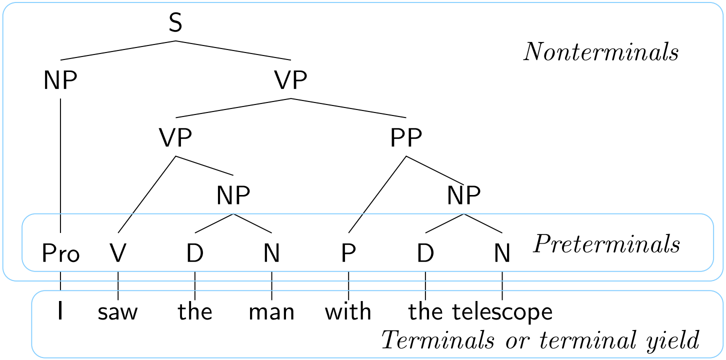
Example of a CF derivation and parse tree

$$\mathcal{R} = \left\{ \begin{array}{ccc} \mathrm{S} \to \mathrm{NP\ VP} & \mathrm{NP} \to \mathrm{D\ N} & \mathrm{VP} \to \mathrm{V} \\ \mathrm{D} \to \mathrm{the} & \mathrm{N} \to \mathrm{dog} & \mathrm{V} \to \mathrm{barks} \end{array} \right\}$$

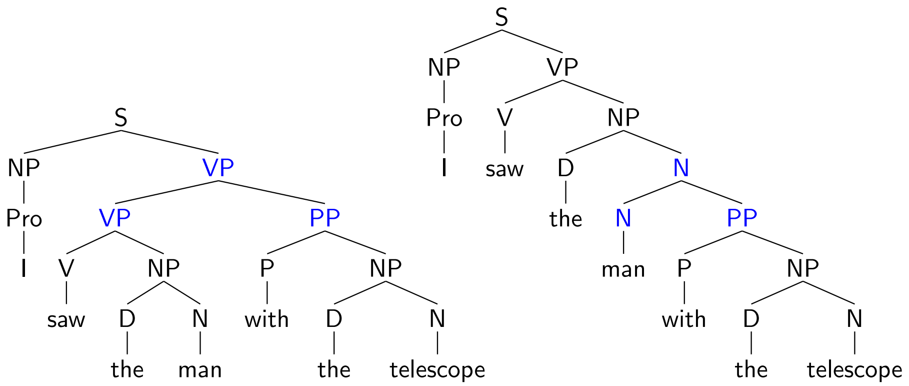


$$\begin{array}{rc} & \mathrm{S} \\ \Rightarrow & \mathrm{NP\ VP} \\ \Rightarrow & \mathrm{D\ N\ VP} \\ \Rightarrow & \mathrm{the\ N\ VP} \\ \Rightarrow & \mathrm{the\ dog\ VP} \\ \Rightarrow & \mathrm{the\ dog\ V} \\ \Rightarrow & \mathrm{the\ dog\ barks} \end{array}$$

Trees can depict constituency



CFGs can describe structural ambiguity

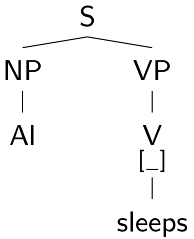


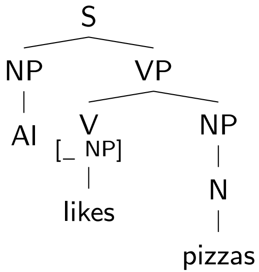
$$\mathcal{R} = \{\mathrm{VP} \rightarrow \mathrm{V\ NP}, \mathrm{VP} \rightarrow \mathrm{VP\ PP}, \mathrm{NP} \rightarrow \mathrm{D\ N}, \mathrm{N} \rightarrow \mathrm{N\ PP}, \ldots\}$$

CFG account of subcategorization

Nouns and verbs differ in the number of *complements* they appear with.
We can use a CFG to describe this by splitting or *subcategorizing* the basic categories.

$$\mathcal{R} = \left\{ \begin{array}{cc} \mathrm{VP} \rightarrow \underset{[\_]}{\mathrm{V}} & \mathrm{VP} \rightarrow \underset{[\_\ \mathrm{NP}]}{\mathrm{V}}\ \mathrm{NP} \\ \underset{[\_]}{\mathrm{V}} \rightarrow \mathrm{sleeps} & \underset{[\_\ \mathrm{NP}]}{\mathrm{V}} \rightarrow \mathrm{likes} \\ \ldots & \ldots \end{array} \right\}$$



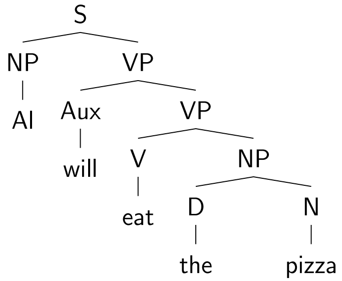


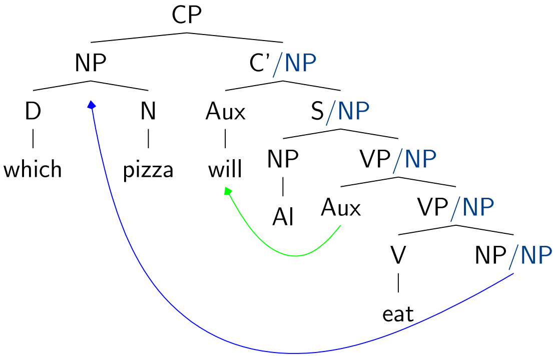
Nonlocal "movement" constructions

"Movement" constructions involve a phrase appearing far from its normal location.

Linguists believed CFGs could not generate them, and posited "movement transformations" to produce them (Chomsky 1957)

But CFGs can generate them via "feature passing" using a conspiracy of rules (Gazdar 1984)





Probabilistic grammars

A probabilistic grammar G defines a probability distribution $\mathrm{P}_G(t)$ over the parse trees $\mathcal{T}$ generated by G, and hence over strings generated by G.

$$\mathrm{P}_G(\boldsymbol{w}) \;=\; \sum_{t \in \mathcal{T}_G(\boldsymbol{w})} \mathrm{P}_G(t)$$

Standard (non-stochastic) grammars distinguish *grammatical* from *ungrammatical* strings (only the grammatical strings receive parses).
Probabilistic grammars can assign non-zero probability to every string, and rely on the probability distribution to distinguish likely from unlikely strings.

MACQUARIE UNIVERSITY

Probabilistic context-free grammars

A *Probabilistic Context Free Grammar* (PCFG) consists of $(\mathcal{W}, \mathcal{N}, S, \mathcal{R}, p)$ where:

- $(\mathcal{W}, \mathcal{N}, S, \mathcal{R})$ is a CFG with *no useless productions or nonterminals*, and
- p is a vector of *production probabilities*, i.e., a function $\mathcal{R} \to [0, 1]$ that satisfies for each $A \in \mathcal{N}$:

$$\sum_{A \to \beta \,\in\, \mathcal{R}(A)} p(A \to \beta) \;=\; 1$$

  where $\mathcal{R}(A) = \{A \to \alpha : A \to \alpha \in \mathcal{R}\}$.

A production $A \to \alpha$ is *useless* iff there are no derivations of the form $S \Rightarrow^\star \gamma A \delta \Rightarrow \gamma \alpha \delta \Rightarrow^* w$ for any $\gamma, \delta \in (\mathcal{N} \cup \mathcal{W})^\star$ and $w \in \mathcal{W}^\star$.

MACQUARIE UNIVERSITY

Probability distribution defined by a PCFG

Intuitive interpretation:

- the probability of rewriting nonterminal A to α is $p(A \to \alpha)$
- the probability of a derivation is the product of probabilities of rules used in the derivation

For each production $A \to \alpha \in \mathcal{R}$, let $f_{A\to\alpha}(t)$ be *the number of times $A \to \alpha$ is used in t*.

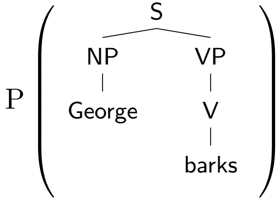
A PCFG G defines a probability distribution P_G on $\mathcal{T}$ that is non-zero on $\mathcal{T}(G)$:

$$\mathrm{P}_G(t) \ = \ \prod_{r \in \mathcal{R}} p(r)^{f_r(t)}$$

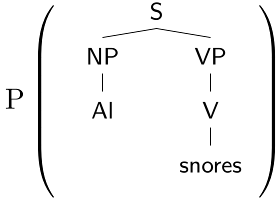
This distribution is properly normalized if p satisfies suitable constraints.

Example PCFG

1.0	$\mathrm{S} \rightarrow \mathrm{NP\ VP}$	1.0	$\mathrm{VP} \rightarrow \mathrm{V}$
0.75	$\mathrm{NP} \rightarrow \mathrm{George}$	0.25	$\mathrm{NP} \rightarrow \mathrm{Al}$
0.6	$\mathrm{VP} \rightarrow \mathrm{barks}$	0.4	$\mathrm{VP} \rightarrow \mathrm{snores}$



$= 0.45$



$= 0.1$

Things we want to compute with PCFGs

Given a PCFG G and a string $\boldsymbol{w} \in \mathcal{W}^\star$,

- (parsing): the most likely tree for $\boldsymbol{w}$,

$$\operatorname*{argmax}_{t \in \mathcal{T}_G(\boldsymbol{w})} \mathrm{P}_G(t)$$

- (language modeling): the probability of $\boldsymbol{w}$,

$$\mathrm{P}_G(\boldsymbol{w}) \;=\; \sum_{t \in \mathcal{T}_G(\boldsymbol{w})} \mathrm{P}_G(t)$$

Learning rule probabilities from data:

- (maximum likelihood estimation from visible data): given a corpus of trees $\boldsymbol{D} = (t_1, \ldots, t_n)$, which rule probabilities p makes $\boldsymbol{D}$ as likely as possible?
- (maximum likelihood estimation from hidden data): given a corpus of strings $\boldsymbol{D} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n)$, which rule probabilities p makes $\boldsymbol{D}$ as likely as possible?

Parsing and language modeling

The probability $\mathrm{P}_G(t)$ of a tree $t \in \mathcal{T}_G(\boldsymbol{w})$ is:

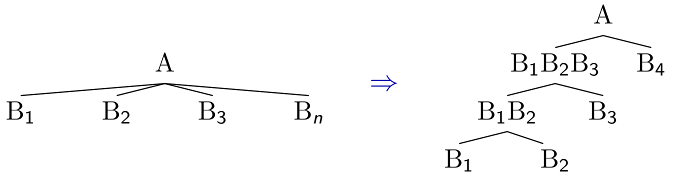
$$\mathrm{P}_G(t) \;=\; \prod_{r \in \mathcal{R}} p(r)^{f_r(t)}$$

Suppose the set of parse trees $\mathcal{T}_G(w)$ is finite, and we can enumerate it.
Naive parsing/language modeling algorithms for PCFG G and string $w \in \mathcal{W}^\star$:

1. Enumerate the set of parse trees $\mathcal{T}_G(\boldsymbol{w})$
2. Compute the probability of each $t \in \mathcal{T}_G(\boldsymbol{w})$
3. Argmax/sum as appropriate

Chomsky normal form

A CFG is in *Chomsky Normal Form* (CNF) iff all productions are of the form $A \to B\ C$ or $A \to x$, where $A, B, C \in \mathcal{N}$ and $x \in \mathcal{W}$.
PCFGs *without epsilon productions* $A \to \epsilon$ can always be put into CNF.
Key step: *binarize* productions with more than two children by introducing new nonterminals



Substrings and string positions

Let $\boldsymbol{w} = w_1 w_2 \dots w_n$ be a string of length n

A *string position* for w is an integer $i \in 0, \dots, n$ (informally, it identifies the position between words w_{i-1} and w_i)

$$\begin{array}{ccccccccc} \bullet & \text{the} & \bullet & \text{dog} & \bullet & \text{chases} & \bullet & \text{cats} & \bullet \\ 0 & & 1 & & 2 & & 3 & & 4 \end{array}$$

A *substring* of $\boldsymbol{w}$ can be specified by beginning and ending string positions

$\boldsymbol{w}_{i:j}$ is the substring starting at word $i+1$ and ending at word j.

$$\boldsymbol{w}_{0:4} = \text{the dog chases cats}$$
$$\boldsymbol{w}_{1:2} = \text{dog}$$
$$\boldsymbol{w}_{2:4} = \text{chases cats}$$

MACQUARIE UNIVERSITY

Language modeling using dynamic programming

- *Goal:* To compute $\mathrm{P}_G(\boldsymbol{w}) = \sum_{t \in \mathcal{T}_G(\boldsymbol{w})} \mathrm{P}_G(t) = \mathrm{P}_G(S \Rightarrow^* \boldsymbol{w})$
- *Data structure:* A table called a *chart* recording $\mathrm{P}_G(A \Rightarrow^* \boldsymbol{w}_{i:k})$ for all $A \in N$ and $0 \leq i < k \leq |\boldsymbol{w}|$
- *Base case:* For all $i = 1, \ldots, n$ and $A \to w_i$, compute:

$$\mathrm{P}_G(A \Rightarrow^* \boldsymbol{w}_{i-1:i}) \;=\; p(A \to w_i)$$

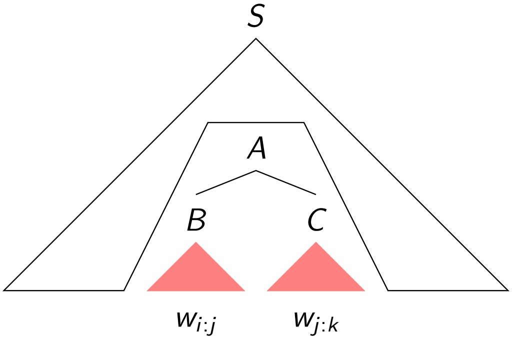
- *Recursion:* For all $k - i = 2, \ldots, n$ and $A \in \mathcal{N}$, compute:

$$\begin{aligned}&\mathrm{P}_G(A \Rightarrow^* \boldsymbol{w}_{i:k}) \\ &\quad = \sum_{j=i+1}^{k-1} \sum_{A \to B\, C \in \mathcal{R}(A)} p(A \to B\, C) \mathrm{P}_G(B \Rightarrow^* \boldsymbol{w}_{i:j}) \mathrm{P}_G(C \Rightarrow^* \boldsymbol{w}_{j:k})\end{aligned}$$

MACQUARIE UNIVERSITY

Dynamic programming recursion

$$\mathrm{P}_G(A \Rightarrow^* \boldsymbol{w}_{i:k}) = \sum_{j=i+1}^{k-1} \sum_{A \to B\,C \in R(A)} p(A \to B\,C)\mathrm{P}_G(B \Rightarrow^* \boldsymbol{w}_{i:j})\mathrm{P}_G(C \Rightarrow^* \boldsymbol{w}_{j:k})$$

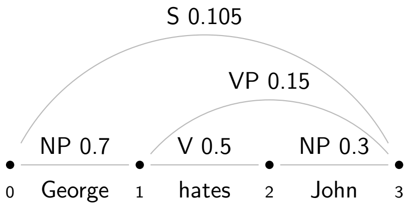


$\mathrm{P}_G(A \Rightarrow^* \boldsymbol{w}_{i:k})$ is called the *inside probability* of A spanning $\boldsymbol{w}_{i:k}$.

Example PCFG string probability calculation

$$w = \text{George hates John}$$

$$R = \left\{ \begin{array}{llll} 1.0 & \text{S} \rightarrow \text{NP VP} & 1.0 & \text{VP} \rightarrow \text{V NP} \\ 0.7 & \text{NP} \rightarrow \text{George} & 0.3 & \text{NP} \rightarrow \text{John} \\ 0.5 & \text{V} \rightarrow \text{likes} & 0.5 & \text{V} \rightarrow \text{hates} \end{array} \right\}$$

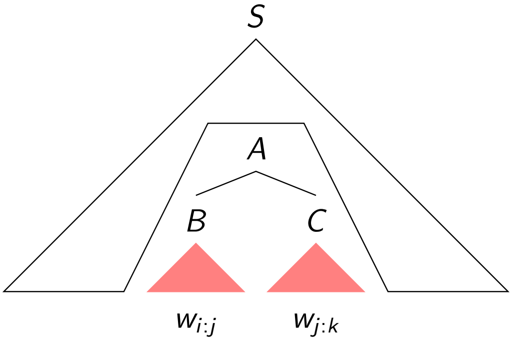


Right string position

Left string position	1	2	3
0	NP 0.7		S 0.105
1		V 0.5	VP 0.15
2			NP 0.3

Computational complexity of PCFG parsing

$$\mathrm{P}_G(A \Rightarrow^* \boldsymbol{w}_{i:k}) \;=\; \sum_{j=i+1}^{k-1} \; \sum_{A \to B\, C \in \mathcal{R}(A)} p(A \to B\; C)\mathrm{P}_G(B \Rightarrow^* \boldsymbol{w}_{i:j})\mathrm{P}_G(C \Rightarrow^* \boldsymbol{w}_{j:k})$$



For each production $r \in \mathcal{R}$ and each i, k, we must sum over all intermediate positions $j \Rightarrow O(n^3 |\mathcal{R}|)$ time

Estimating (learning) PCFGs from data

Estimating productions and production probabilities from *visible data* (corpus of parse trees) is straight-forward:

- the productions are identified by the local trees in the data
- *Maximum likelihood principle:* select production probabilities in order to make corpus as likely as possible

Estimating production probabilities from *hidden data* (corpus of terminal strings) is much more difficult:

- The *Expectation-Maximization* (EM) algorithm finds probabilities that *locally maximize* likelihood of corpus
- The *Inside-Outside* algorithm runs in cubic time in length of corpus

Estimating PCFGs from visible data

Data: A *treebank* of parse trees $D = (t_1, \ldots, t_n)$.

$$L_D(p) \;=\; \prod_{i=1}^{n} \mathrm{P}_G(t_i) \;=\; \prod_{A \to \alpha \in \mathcal{R}} p(A \to \alpha)^{f_{A\to\alpha}(D)}$$

where $f_{A\to\alpha}(D) = \sum_{i=1}^{n} f_{A\to\alpha}(t_i)$ is the number of times $A \to \alpha$ is used in D.
Introduce $|N|$ Lagrange multipliers c_B for each $B \in \mathcal{N}$ for the constraints $\sum_{B\to\beta\in\mathcal{R}(B)} p(B \to \beta) = 1$:

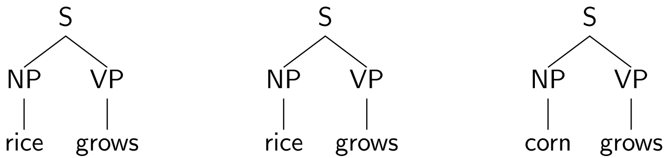
$$\frac{\partial \left(L_D(p) - \sum_{B\in\mathcal{N}} c_B \left(\sum_{B\to\beta\in\mathcal{R}(B)} p(B\to\beta) - 1 \right) \right)}{\partial p(A \to \alpha)} \;=\; \frac{L_D(p) f_r(D)}{p(A\to\alpha)} - c_A$$

Setting this to 0, $\quad p(A \to \alpha) \;=\; \dfrac{f_{A\to\alpha}(D)}{\sum_{A\to\alpha'\in\mathcal{R}(A)} f_{A\to\alpha'}(D)}$

MACQUARIE UNIVERSITY

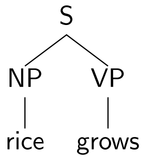
Visible PCFG estimation example

D =



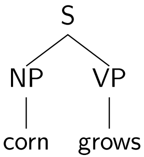
Rule	**Count**	**Rel Freq**
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	$2/3$
NP $\rightarrow$ corn	1	$1/3$
VP $\rightarrow$ grows	3	1

P (



) $= 2/3$

P (



) $= 1/3$

Estimating production probabilities from hidden data

Data: A corpus of sentences $D = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n)$.

$$L_D(p) = \prod_{i=1}^{n} \mathrm{P}_G(\boldsymbol{w}_i). \qquad \mathrm{P}_G(\boldsymbol{w}) = \sum_{t \in \mathcal{T}_G(w)} \mathrm{P}_G(t).$$

$$\frac{\partial L_D(p)}{\partial p(A \to \alpha)} = \frac{L_D(p) \sum_{i=1}^{n} \mathrm{E}_G[f_{A\to\alpha} | \boldsymbol{w}_i]}{p(A \to \alpha)}$$

After introducing a Lagrange multiplier for the constraint $\sum_{B\to\beta\in\mathcal{R}(B)} p(B \to \beta) = 1$:

$$p(A \to \alpha) = \frac{\sum_{i=1}^{n} \mathrm{E}_G[f_{A\to\alpha} \mid \boldsymbol{w}_i]}{\sum_{A\to\alpha'\in\mathcal{R}(A)} \sum_{i=1}^{n} \mathrm{E}_G[f_{A\to\alpha'} \mid \boldsymbol{w}_i]}$$

This is an iteration of the *Expectation Maximization* algorithm!

MACQUARIE UNIVERSITY

The EM algorithm for PCFGs

Input: a corpus of strings $D = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n)$
Guess initial production probabilities $p^{(0)}$
For $u = 1, 2, \ldots$ do:

1. Calculate *expected frequency* $\sum_{i=1}^{n} \mathrm{E}_{p^{(u-1)}}[f_{A\to\alpha}|\boldsymbol{w}_i]$ of each production:

$$\mathrm{E}_p[f_{A\to\alpha} \mid \boldsymbol{w}] \quad = \quad \sum_{t \in \mathcal{T}_G(w)} f_{A\to\alpha}(t)\mathrm{P}_p(t)$$

2. Set $p^{(u)}$ to the *relative expected frequency* of each production

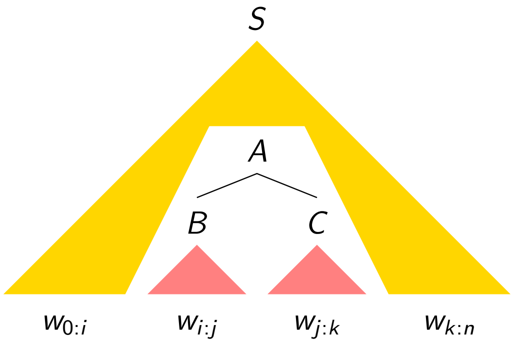
$$p^{(u)}(A \to \alpha) \quad = \quad \frac{\sum_{i=1}^{n} \mathrm{E}_{p^{(u-1)}}[f_{A\to\alpha}|\boldsymbol{w}_i]}{\sum_{A\to\alpha'} \sum_{i=1}^{n} \mathrm{E}_{p^{(u-1)}}[f_{A\to\alpha'}|\boldsymbol{w}_i]}$$

It is as if $p^{(u)}$ were estimated from a visible corpus $\mathcal{T}_G$ in which each tree t occurs $\sum_{i=1}^{n} \mathrm{P}_{p^{(u-1)}}(t|\boldsymbol{w}_i)$ times.

Dynamic programming for $E_p[f_{A \to B\ C} \mid \boldsymbol{w}]$

$E_p[f_{A \to B\ C} \mid \boldsymbol{w}] =$

$$\frac{\displaystyle\sum_{0 \leq i < j < k \leq n} \mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{1:i}\, A\, \boldsymbol{w}_{k:n}) p(A \to B\ C) \mathrm{P}(B \Rightarrow^* \boldsymbol{w}_{i:j}) \mathrm{P}(C \Rightarrow^* \boldsymbol{w}_{j:k})}{\mathrm{P}_G(\boldsymbol{w})}$$



Calculating “outside probabilities”

Construct a table of “outside probabilities” $\mathrm{P}_G(S \Rightarrow^* \boldsymbol{w}_{0:i}\, A\, \boldsymbol{w}_{k:n})$ for all $0 \leq i < k \leq n$ and $A \in \mathcal{N}$

Recursion from *larger to smaller* substrings in $\boldsymbol{w}$.

Base case: $\mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:0}\, S\, \boldsymbol{w}_{n:n}) = 1$

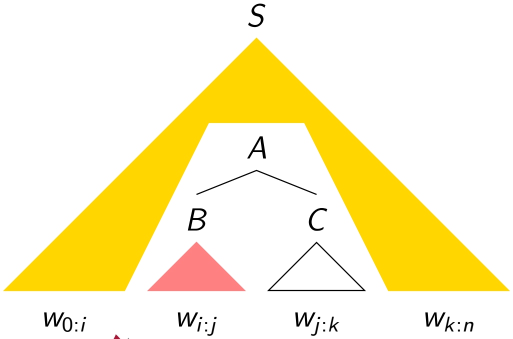
Recursion: $\mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:j}\, C\, \boldsymbol{w}_{k:n}) =$

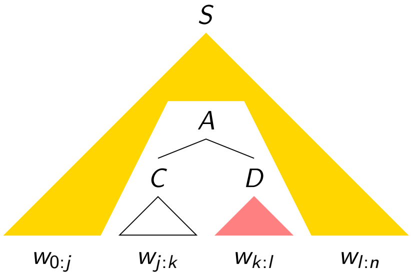
$$\sum_{i=0}^{j-1} \sum_{\substack{A,B \in \mathcal{N} \\ A \to B\ C \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:i}\, A\, \boldsymbol{w}_{k:n}) p(A \to B\ C) \mathrm{P}(B \Rightarrow^* \boldsymbol{w}_{i:j})$$

$$+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{N} \\ A \to C\ D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:j}\, A\, \boldsymbol{w}_{l:n}) p(A \to C\ D) \mathrm{P}(D \Rightarrow^* \boldsymbol{w}_{k:l})$$

Recursion in $\mathrm{P}_G(S \Rightarrow^* \boldsymbol{w}_{0:i}\, A\, \boldsymbol{w}_{k:n})$

$$\begin{aligned}
\mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:j}\, C\, \boldsymbol{w}_{k:n}) = \\
& \sum_{i=0}^{j-1} \sum_{\substack{A,B \in \mathcal{N} \\ A \to B\ C \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:i}\, A\, \boldsymbol{w}_{k:n}) p(A \to B\ C) \mathrm{P}(B \Rightarrow^* \boldsymbol{w}_{i:j}) \\
+ & \sum_{l=k+1}^{n} \sum_{\substack{A,D \in \mathcal{N} \\ A \to C\ D \in \mathcal{R}}} \mathrm{P}(S \Rightarrow^* \boldsymbol{w}_{0:j}\, A\, \boldsymbol{w}_{l:n}) p(A \to C\ D) \mathrm{P}(D \Rightarrow^* \boldsymbol{w}_{k:l})
\end{aligned}$$





Expectation Maximization with a toy grammar

Initial rule probs

rule	prob
...	...
VP → V	0.2
VP → V NP	0.2
VP → NP V	0.2
VP → V NP NP	0.2
VP → NP NP V	0.2
...	...
Det → the	0.1
N → the	0.1
V → the	0.1

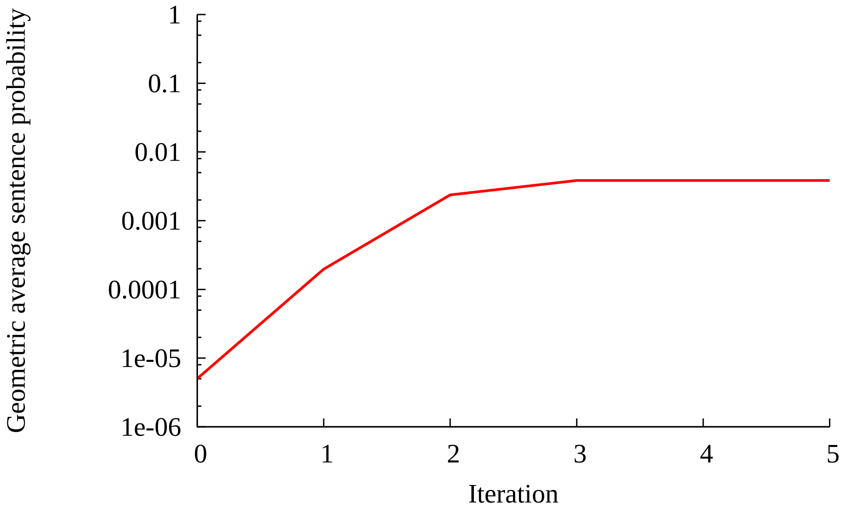
"English" input

the dog bites
the dog bites a man
a man gives the dog a bone
...

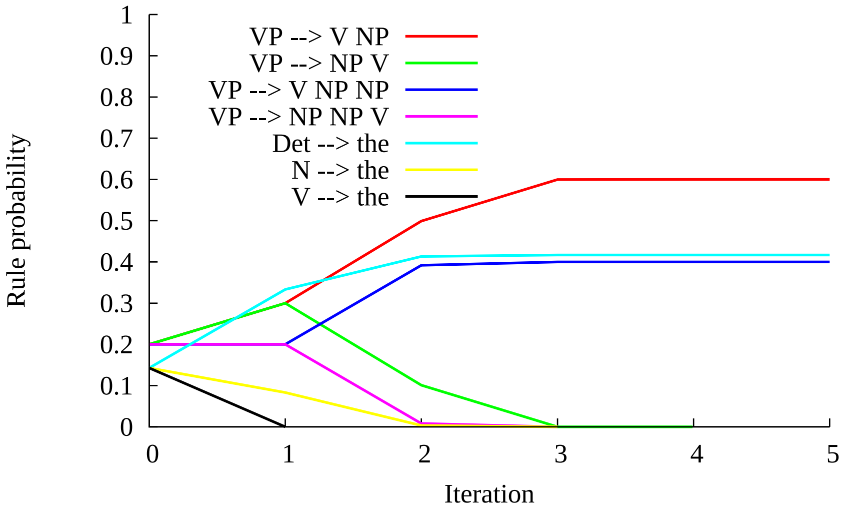
"pseudo-Japanese" input

the dog bites
the dog a man bites
a man the dog a bone gives
...

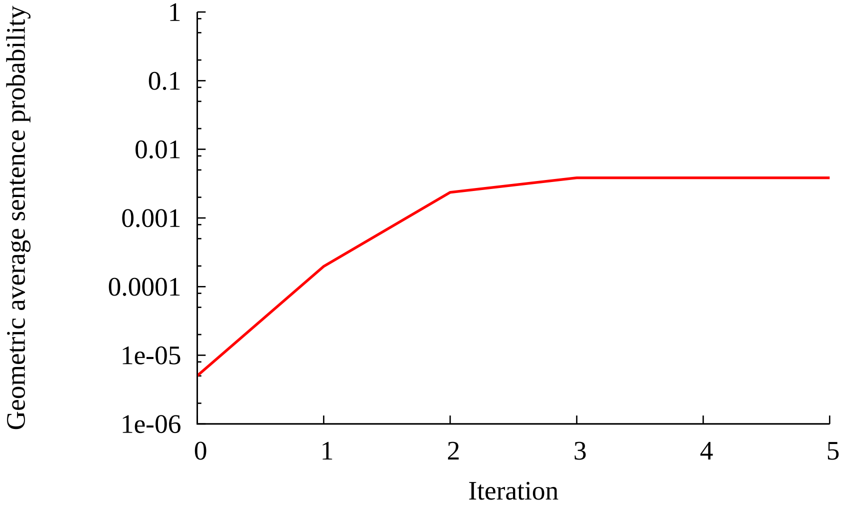
Probability of “English”



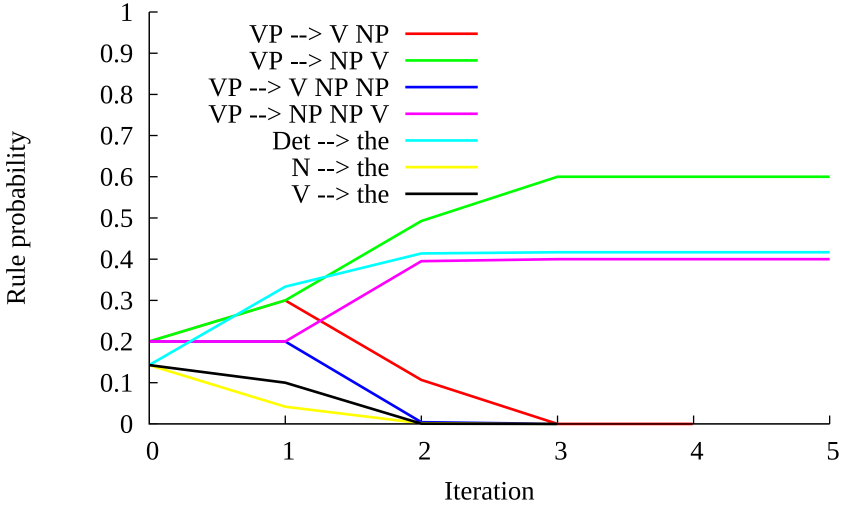
Rule probabilities from “English”



Probability of “Japanese”

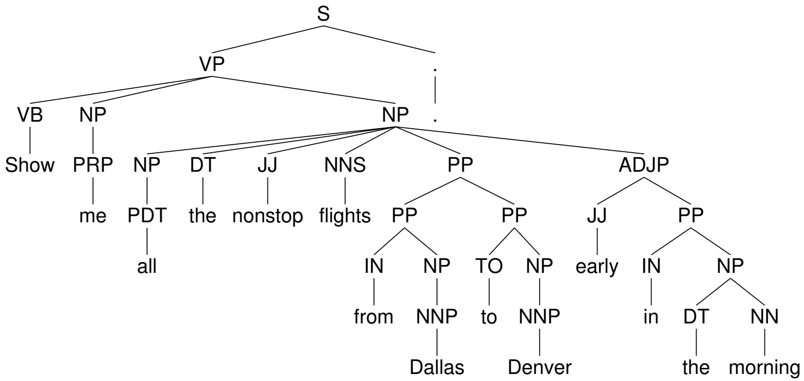


Rule probabilities from “Japanese”

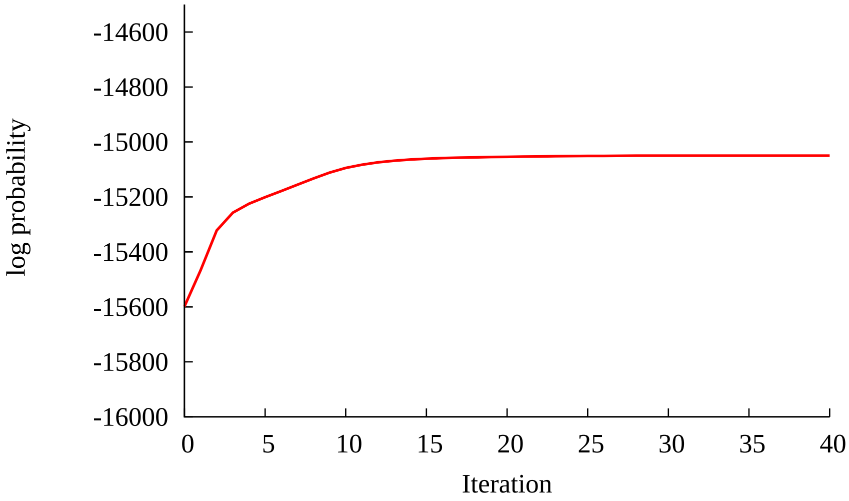


Expectation Maximisation on real text

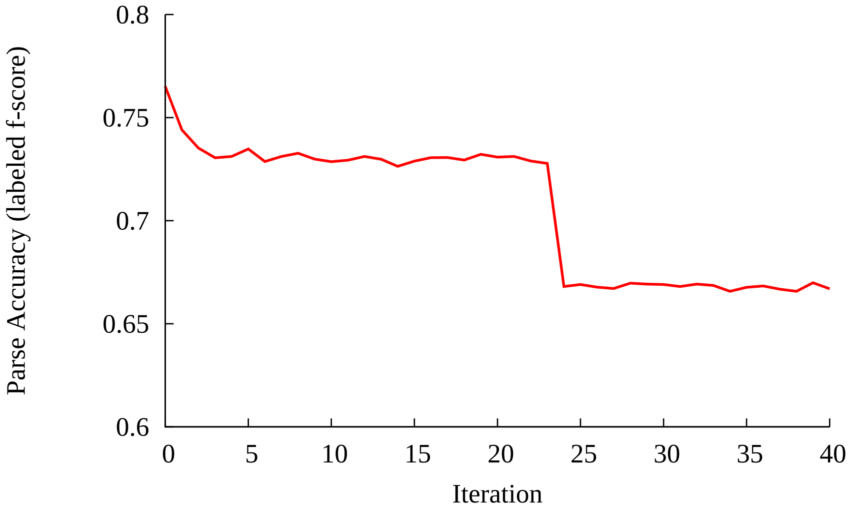
- ATIS treebank consists of 1,300 hand-constructed parse trees
- input consists of POS tags rather than words
- about 1,000 PCFG rules are needed to build these trees



Probability of training strings



Accuracy of parses of training strings



Summary of PCFG parsing

- PCFGs define probability distributions over trees
  - Context free $\Rightarrow$ local dependencies $\Rightarrow$ efficient dynamic programming
- Expectation Maximisation on its own can't learn grammars from scratch
  - it is able to learn specialised grammars
  - most work on unsupervised grammar induction focuses on dependency grammars

Outline

What is parsing?

- Syntactic structure depicts:
    - the way words combine to form phrases and sentences, and
    - the dependencies between these words and phrases
- A grammar is a finite specification of the syntactic structures of a language
    - a *probabilistic grammar* defines a probability distribution over syntactic structures
- Parsing is the process of recovering syntactic structures from text or speech

Parsing lies on the path from form to meaning

- "Classical AI": map text/speech to "meaning representations"
- Might be able to do this if only we knew what "meaning" is
- Problem is figuring out
  - what information should be in a "meaning representation", and
  - what to do with a meaning representation once you have it

$$\begin{tikzcd}
\text{Text or speech} \arrow[d, Rightarrow] \\
\text{Syntactic parse} \arrow[d, Rightarrow] \\
\text{Meaning representation}
\end{tikzcd}$$

Two kinds of uses for probabilistic parsing

"If you build it, they will come . . ."

- Parses as (proxy) meaning representations
  - machine translation
    English text → *English parse* → *French parse* → *French text*
  - text summarization
  - information retrieval / question answering
    parsing may become more useful with higher bandwidth inputs and lower bandwidth outputs

  (should optimize information parses contain for each application)
- Improved language models for "noisy channel" applications
  (use parsing model to compute string probability; throw away parse trees)
  - speech recognition
  - machine translation

Linguistic grammar-based approaches to parsing

- Division of labour:
  - Computational linguists
    - design a formalism for stating grammars
      (e.g., Lexical-functional grammar (LFG),
      Head-driven phrase-structure grammar (HPSG),
      Combinatory Categorial Grammar (CCG))
    - design algorithms that find parses generated by *any grammar* written in formalism
  - Linguists write grammar fragments in formalism that describe interesting constructions
- Central questions:
  - how complicated does the formalism for stating grammars need to be?
  - how much computational work is required to *parse*, i.e., find the analysis of a sentence

The ambiguity vs. coverage vs. detail trilemma

- *Ambiguity* grows exponentially with length of sentence

  Put the rice in a box on the table in the kitchen . . .
  - world-knowledge and pragmatics might resolve some ambiguities
  - many syntactic ambiguities don't correspond to differences in meaning
- *Coverage:* no complete grammars of English or any other language
  - as you increase coverage, ambiguity usually increases as well
- *Detail:* calling in the linguists often makes matters worse
  - E.g., systematically distinguishing count and mass nouns adds more ambiguity than it resolves

Statistical approaches to parsing

- Goal: find *most probable analysis* of a string of words
- Approach:
  - define a probability distribution $\mathrm{P}(\mathrm{Tree})$ over *all possible analyses of all possible strings*
  - from this, we can compute:
    - probability of a tree given words $\mathrm{P}(\mathrm{Tree} \mid \mathrm{Words})$
    - probability that a sequence of words is a sentence $\mathrm{P}(\mathrm{Words})$ (*language model*)
    - probability that a sequence of words is a *prefix* of a sentence
- Doesn't decide whether an analysis Tree or a string Words is grammatical
  - usually assign every possible Tree a positive probability
    $\Rightarrow$ guarantees *every string gets an analysis*
  - but these probabilities can *vary enormously*

A new division of labour

- Why has the statistical approach been so successful?
- Factor the parsing task into two steps:
  - linguists design (and annotate) the corpora
  - computer scientists design machine-learning algorithms that generalise from the corpora
- Each group needs only minimal understanding of the other's task

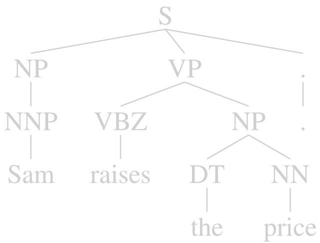
Conventional grammars are closed-world

- The *closed world assumption* for grammars
  - Rules and lexical entries define set of grammatical structures
  - Everything not grammatical is ungrammatical
- "Parsing as deduction": parsing is the process of proving the grammaticality of a sentence
- But: goal is *understanding* what the speaker's trying to say; not determining whether the sentence is grammatical
- Ideal parser should be *open-world*
  - even ungrammatical sentences are interpretable
    E.g., *man bites dog* $\neq$ *dog bites man*
  - words and constructions we recognize provide information about sentence's meaning
  - unknown words or phrases do not cause interpretation to fail
    - parsing and acquisition are two aspects of same process
- *Statistical treebank parsers are open-world*
  - every possible tree receives positive probability

MACQUARIE UNIVERSITY

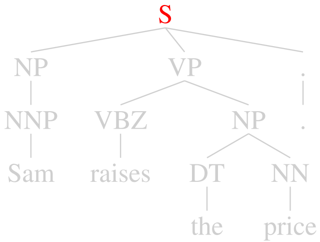
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



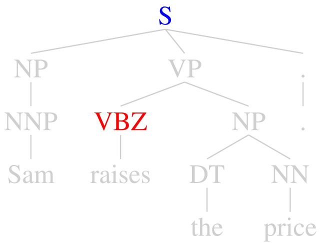
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



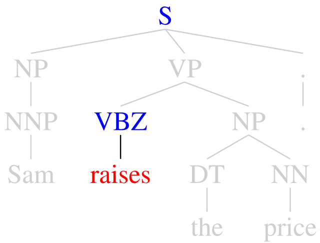
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



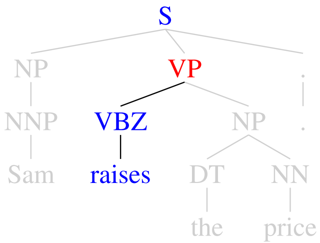
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



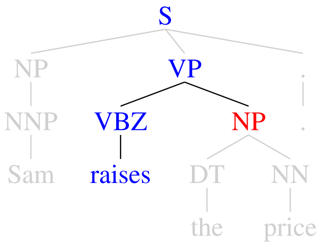
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



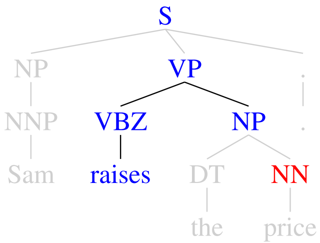
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
    - governor and its maximal projection
    - governor's governor and its maximal projection



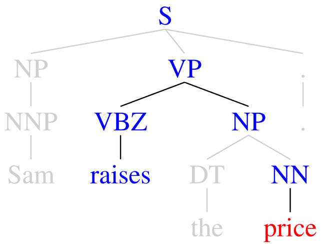
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
    - governor and its maximal projection
    - governor's governor and its maximal projection



A “history-based” generative parsing model

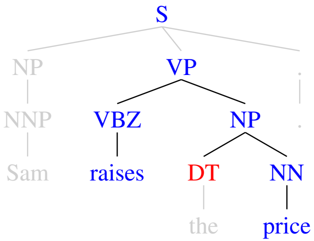
- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor’s governor and its maximal projection



MACQUARIE UNIVERSITY

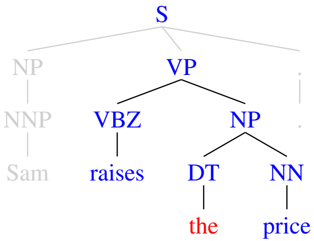
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



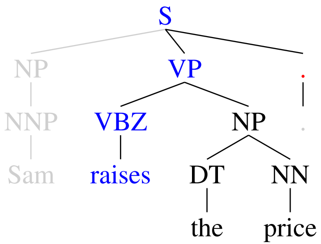
A “history-based” generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
    - governor and its maximal projection
    - governor’s governor and its maximal projection



A “history-based” generative parsing model

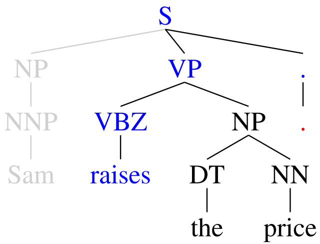
- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor’s governor and its maximal projection



MACQUARIE UNIVERSITY

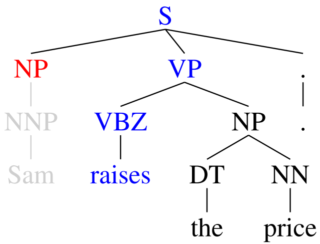
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



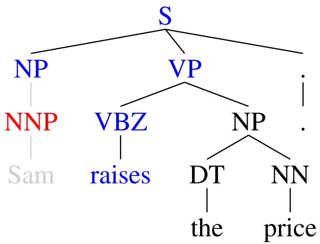
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
    - governor and its maximal projection
    - governor's governor and its maximal projection



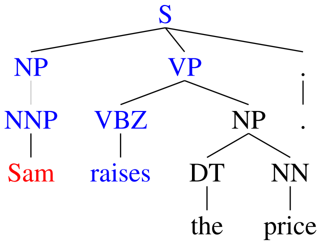
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



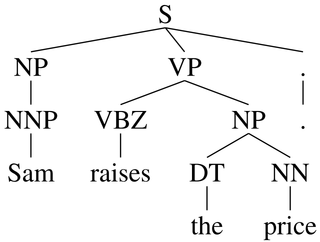
A “history-based” generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
    - governor and its maximal projection
    - governor's governor and its maximal projection



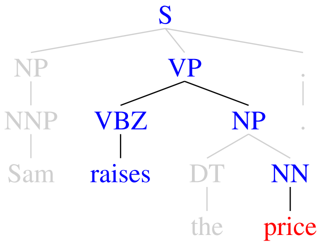
A "history-based" generative parsing model

- Generates nodes of tree in (roughly) top-down order
- *Predict each node from nodes already generated:*
  - governor and its maximal projection
  - governor's governor and its maximal projection



Estimating grammar probabilities from treebanks

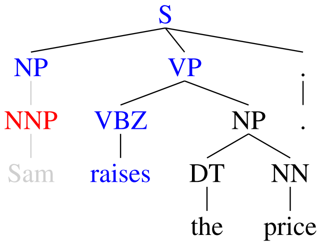
- The statistical parser needs to know the conditional probability of generating node type at each location in the tree
- A *treebank* is a corpus in which each sentence is labelled with its analysis tree



- The Penn WSJ treebank contains parses for $\sim$ 40,000 sentences (1.2 million words)
- The conditional probabilities can be estimated from a treebank by *counting how often each node type appears in each context*
- There are methods for learning these probabilities from strings alone, but the resulting grammars are not very good

Capturing vs. covering linguistic generalizations

- *Capturing a generalization*: grammar accurately describes phenomenon at appropriate level, e.g., subject-verb agreement via PERSON and NUMBER features
- *Covering a generalization*: model covers common cases of a generalization, perhaps indirectly. E.g., head-to-head POS dependencies



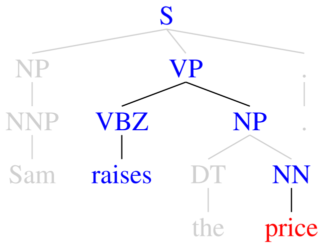
- An "engineering" parser only needs to cover generalizations
- Even so, feature design requires *linguistic insight*
  - basic linguistic insights are often most useful

Parsing in the late 1990s

- Parsers for hand-written grammars (LFG, HPSG, etc)
    - linguistically rich, detailed representations
    - uneven / poor coverage of English
    - even simple sentences are highly ambiguous
    - only ad hoc treatment of preferences
    - could not be learnt from data
- Generative probabilistic parsers
    - systematic treatment of preferences
    - *learnt from treebank corpora*
    - simple constituent structure representations
    - wide (if superficial) coverage of English
- *Could the two approaches be combined?*

Generative statistical parsers

- "History-based" generative statistical parsers (Bikel, Charniak, Collins) generate each new node in parse conditioned on the structure already generated:



$$P(\text{price} | \text{NN}, \text{NP}, \text{raises}, \text{VBZ}, \text{VP}, \text{S})$$

- Assume each node is independent of all existing structure except for nodes explicitly conditioned on

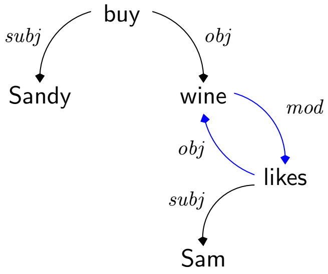
$\Rightarrow$ $P(t)$ is product of node probabilities

$\Rightarrow$ simple "relative frequency" estimators (smoothing is essential $\Diamond\Diamond$)

Reentrant dependencies cause problems

- Linguistic structures exhibit *cyclic dependencies*
  - Linguistically-motivated grammars typically include many kinds of these
- Cannot generate such parses as sequence of independent steps (naive approach "double counts")

⇒ Estimating parameters of such grammars is computationally much harder



"Sandy bought the wine Sam likes"

Generalising beyond generative models

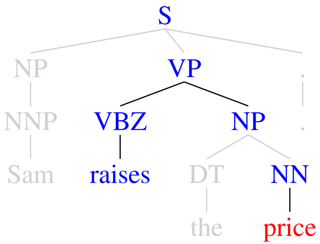
- Mathematically straight-forward to define models in which parse is is not generated as a sequence of independent decisions
  - Maximum Entropy, log-linear, exponential, Gibbs, ...

$$\mathrm{P}(t) \quad = \quad \frac{1}{Z} \exp \sum_{j=1}^{m} w_j f_j(t)$$

- Once we no longer require factors to be independent
  - parses t need not be trees
  - features f_j can be *any computable function* of parses
    e.g., one feature for each possible kind of dependency
- But simple "relative frequency" estimators for feature weights w no longer work; maximum likelihood estimation is very hard

Abney (1997)

Log-linear models are more expressive



- Define features $f_{x,c}$ for all node labels x and contexts c

  $f_{(\text{price},\text{NN},\text{NP},\text{raises},\text{VBZ},\text{VP},\text{S})}(t)$ is the number of times $(\text{price}, \text{NN}, \text{NP}, \text{raises}, \text{VBZ}, \text{VP}, \text{S})$ appears in parse t
- Let weight $w_{x,c} = \log \mathrm{P}(x|c)$

  $w_{(\text{price},\text{NN},\text{NP},\text{raises},\text{VBZ},\text{VP},\text{S})} = \log \mathrm{P}(\text{price}|\text{NN}, \text{NP}, \text{raises}, \text{VBZ}, \text{VP}, \text{S})$
- Generative and log-linear models define same prob. distribution
- Easy to add additional features to log-linear model, but difficult to add additional features to the generative model

Why is maximum likelihood estimation hard?

- Given treebank training corpus $D = (t_1^\star, \ldots, t_n^\star)$, *maximum likelihood estimator* adjusts feature weights w to maximize $\mathrm{P}(D) = \prod_i \mathrm{P}(t_i^\star)$

$$\begin{aligned}\mathrm{P}(t) &= \frac{1}{Z} \exp \sum_{j=1}^{m} w_j f_j(t) \\ Z &= \sum_{t' \in \mathcal{T}} \exp \sum_{j=1}^{m} w_j f_j(t')\end{aligned}$$

  where $\mathcal{T}$ is set of *all possible trees for all possible strings*
- ML estimator adjusts $\boldsymbol{w}$ to make trees in D more likely than any other trees
- Features form a tree of independent generative steps $\Rightarrow$ $Z = 1$ $\Rightarrow$ estimation easy
- Features have complex dependencies $\Rightarrow$ $Z \neq 1$ $\Rightarrow$ estimation hard

MACQUARIE UNIVERSITY

Conditional maximum likelihood estimation

- *Most likely parse* $\hat{t}(s)$ for string s only depends on $\mathrm{P}(t|s)$

⇒ only need to estimate $\mathrm{P}(t|s)$

$$\hat{t}(s) = \operatorname*{argmax}_{t \in \mathcal{T}(s)} \mathrm{P}(t|s), \text{where } \mathcal{T}(s) = \text{set of possible trees } \textit{for } s$$

$$\mathrm{P}(t|s) = \frac{1}{Z(s)} \exp \sum_j w_j f_j(t); \quad Z(s) = \sum_{t \in \mathcal{T}(s)} \exp \sum_j w_j f_j(t)$$

- Partition functions $Z(s_i)$ "only" require summing over $\mathcal{T}(s_i)$, i.e., all parses for each string s_i in training data D
- *Conditional* maximum likelihood selects $\boldsymbol{w}$ to optimize conditional probability of training data $D = ((s_1, t_1^\star), \ldots, (s_n, t_n^\star))$

$$\hat{w} = \operatorname*{argmax}_{w} \prod_{i=1}^{n} \mathrm{P}(t_i^\star | s_i)$$

Johnson, Geman, Canon, Chi and Riezler (1999)

Conditional estimation

s_i	$f(t_i^\star)$	feature vectors of other parses for s_i
sentence 1	$(1,3,2)$	$(2,2,3)$ $(3,1,5)$ $(2,6,3)$
sentence 2	$(7,2,1)$	$(2,5,5)$
sentence 3	$(2,4,2)$	$(1,1,7)$ $(7,2,1)$
...	...	...

- Treebank training data D provides correct parse $t_i^\star$ for sentence s_i
- Parser produces all possible parse trees for each sentence s
- Feature extractor extracts feature vectors $(f_1(t), \ldots, f_m(t))$ for each parse tree t
- Estimator selects feature weights $w = (w_1, \ldots, w_m)$ to make each $t_i^\star$ score as high as possible relative to other parses for s_i

Conditional vs joint estimation

$$\mathrm{P}(t, s) \quad = \quad \mathrm{P}(t|s)\mathrm{P}(s)$$

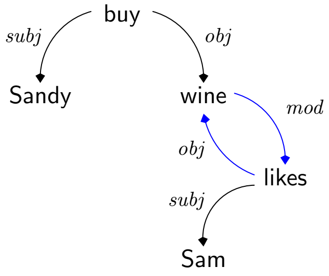
- MLE maximizes probability of training trees $\boldsymbol{t}^\star$ *and strings* $\boldsymbol{s}$
  - Can be used for *language modeling* $\mathrm{P}(s) = \sum_{t \in \mathcal{T}(s)} \mathrm{P}(t, s)$
  - Extends to unsupervised estimation via EM
- Conditional MLE maximizes probability of $\boldsymbol{t}^\star$ *given strings* $\boldsymbol{s}$
  - estimates exactly what we need for parsing
  - uses less information from the data (ignores $\mathrm{P}(s)$)
  - ignores unambiguous sentences (i.e., $|\mathcal{T}(s)| = 1$)
- Joint estimation should be better (lower variance) if model correctly relates $\mathrm{P}(t|s)$ and $\mathrm{P}(s)$
- Conditional estimation should be better if model incorrectly relates $\mathrm{P}(t|s)$ and $\mathrm{P}(s)$

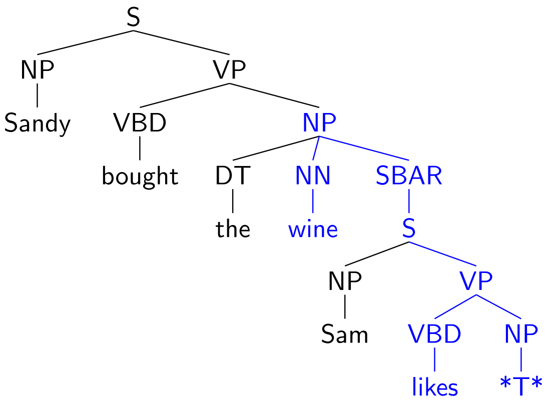
MACQUARIE UNIVERSITY

Details of syntactic representation don't matter

- Log-linear models make decisions based only on *feature vectors* $f(t)$

$\Rightarrow$ Representation t doesn't matter as long as t is "rich enough" to define $f_j(t)$





Conditional estimation for parse rescoring

- Easy to over-fit training data with large number of features

$\Rightarrow$ Regularize by adding a penalty term to log likelihood
  - L1 penalty term $\Rightarrow$ sparse feature weight vector
  - L2 penalty term (Gaussian prior) seems best

- 50-best parses $\mathcal{T}(s)$ may not include true parse $t^\star(s)$

$\Rightarrow$ Train rescorer to prefer parse in $\mathcal{T}_c(s)$ closest to $t^\star(s)$

- Often several parses from 50-best list are equally close to true parse

$\Rightarrow$ EM-inspired (non-convex) loss function

- Direct numerical optimization with L-BFGS (modified for L1 regularizer) produces best results

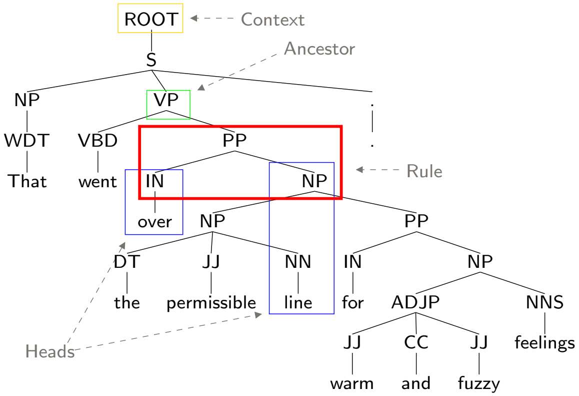
Riezler, King, Kaplan, Crouch, Maxwell and Johnson (2002),
Goodman (2004), Andrew and Gao (2007)

Features for rescoring parses

- Parse rescorer's features can be *any computable function* of parse
- Choice of features is the most important and least understood aspect of the parser
  - feature design has a much greater impact on performance than the learning algorithm
- Features can be based on a linguistic theory (or more than one)
- ...but need not be
  - "shot-gun" or "hail Mary" features very useful
- Feature selection: a feature's values on $t^\star(s)$ and $\mathcal{T}(s) \setminus \{t^\star(s)\}$ must differ on at least 5 sentences $s \in D$
- The *Charniak parser's log probability* combines all of the generative parser's conditional distributions into a single rescorer feature
  $\Rightarrow$ rescoring should never hurt

Lexicalized and parent-annotated rules

Lexicalization associates each constituent with its head
Ancestor annotation provides a little "vertical context"
Context annotation indicates constructions that only occur in main clause (c.f., Emonds)



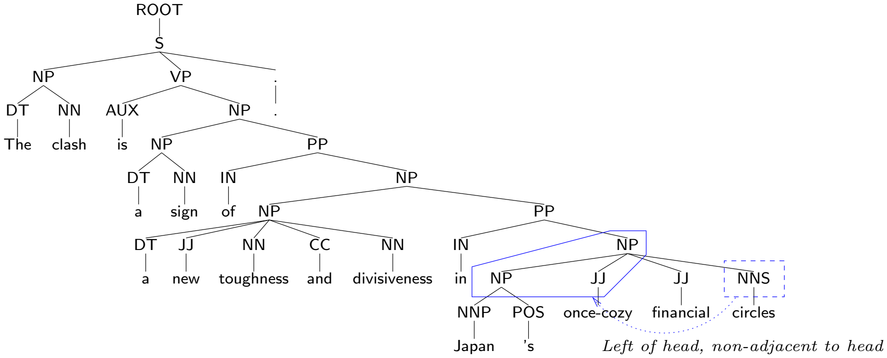
n-gram rule features generalize rules

Collects *adjacent constituents* in a local tree

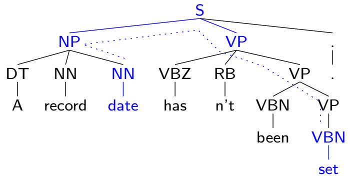
Also includes *relationship to head* (e.g., adjacent? left or right?)

Parameterized by *ancestor-annotation*, *lexicalization* and *head-type*

There are 5,143 unlexicalized rule bigram features and 43,480 lexicalized rule bigram features



Bihead dependency features



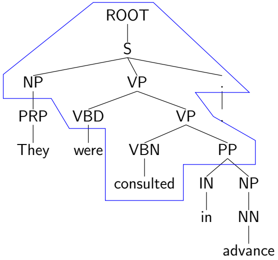
(S (NP (NN date)) (VP (VBN set)))

- Bihead dependency features approximate linguistic function-argument dependencies
- Computed for lexical (≈ semantic) and functional (≈ syntactic) heads
- One feature for each head-to-head dependency found in training corpus (70,000 features in all)

MACQUARIE UNIVERSITY

Head trees record all dependencies

- Head trees consist of a (lexical) head, all of its projections and (optionally) all of the siblings of these nodes
- correspond roughly to TAG elementary trees
- parameterized by *head type*, *number of sister nodes* and *lexicalization*

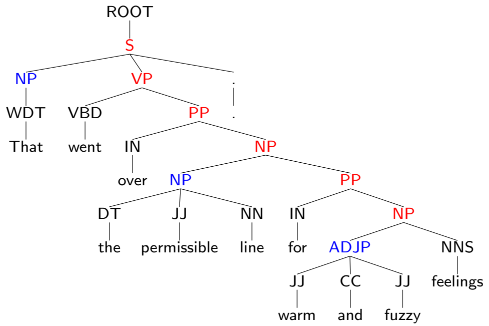


Rightmost branch bias

The RightBranch feature's value is the number of nodes on the right-most branch (ignoring punctuation) (c.f., Charniak 00)

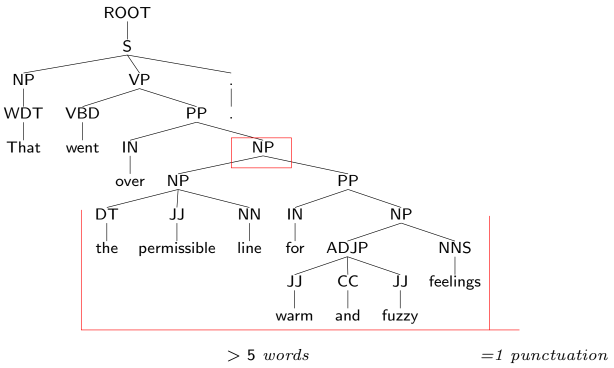
Reflects the tendancy toward right branching in English

Only 2 different features, but very useful in final model!



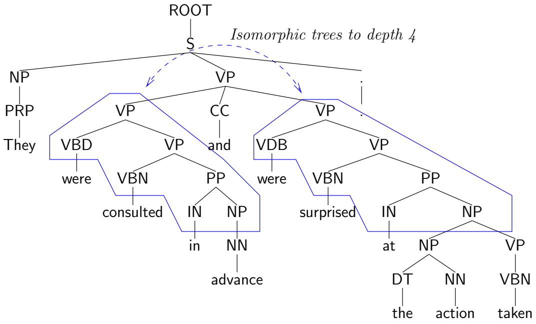
Constituent Heavyness and location

- Heavyness measures the constituent's category, its (binned) size and (binned) closeness to the end of the sentence



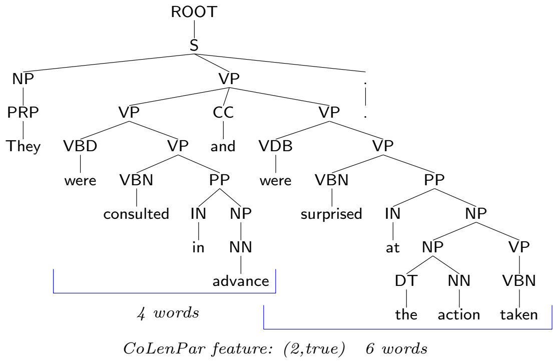
Coordination parallelism (1)

- A CoPar feature indicates the depth to which adjacent conjuncts are parallel



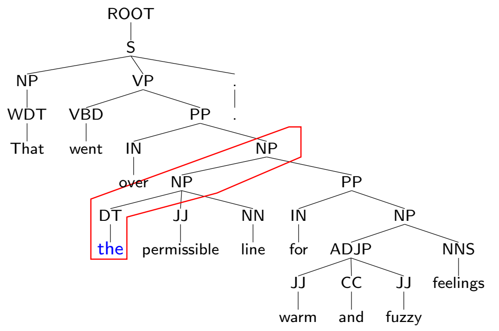
Coordination parallelism (2)

- The CoLenPar feature indicates the difference in length in adjacent conjuncts and whether this pair contains the last conjunct.

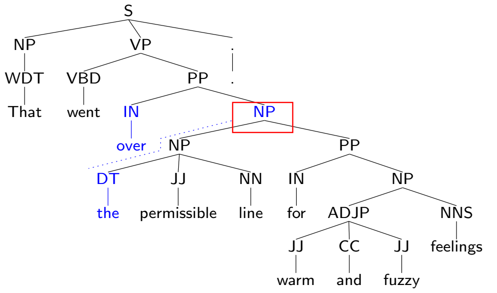


Word

- A Word feature is a word plus *n* of its parents (c.f., Klein and Manning's non-lexicalized PCFG)
- A WProj feature is a word plus all of its (maximal projection) parents, up to its governor's maximal projection



Constituent “edge neighbour” features

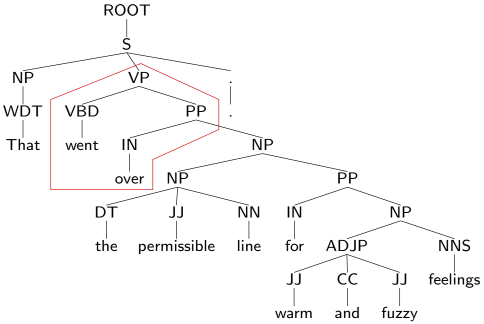


(IN over) (NP (DT the . . .))

- Edge features are a kind of bigram context for constituents
- Would be difficult to incorporate into a generative parser

Tree *n*-gram

- A tree *n*-gram feature is a tree fragment that connect sequences of adjacent *n* words, for $n = 2, 3, 4$ (c.f. Bod's DOP models)
- lexicalized and non-lexicalized variants
- There are 62,487 tree *n*-gram features



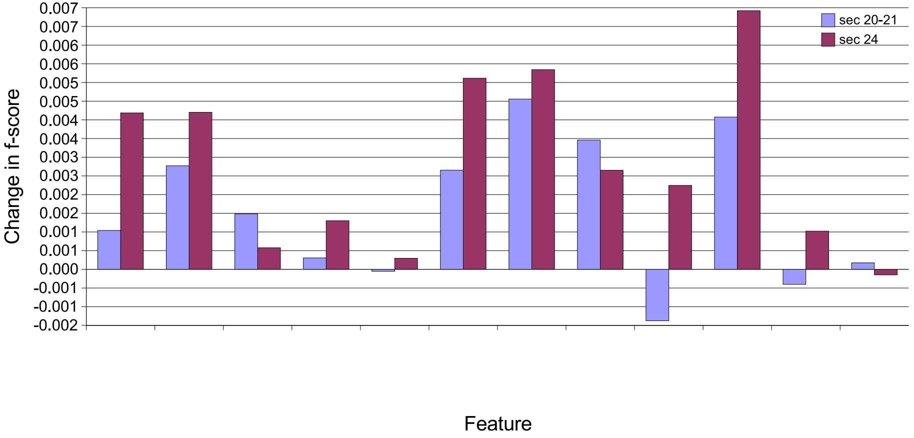
Experimental setup

- Feature tuning experiments done using Collins' split: sections 2-19 as train, 20-21 as dev and 24 as test
- $\mathcal{T}_c(s)$ computed using Charniak 50-best parser
- Features which vary on less than 5 sentences pruned
- Optimization performed using LMVM optimizer from Petsc/TAO optimization package or Averaged Perceptron
- Regularizer constant c adjusted to maximize f-score on dev

Evaluating features

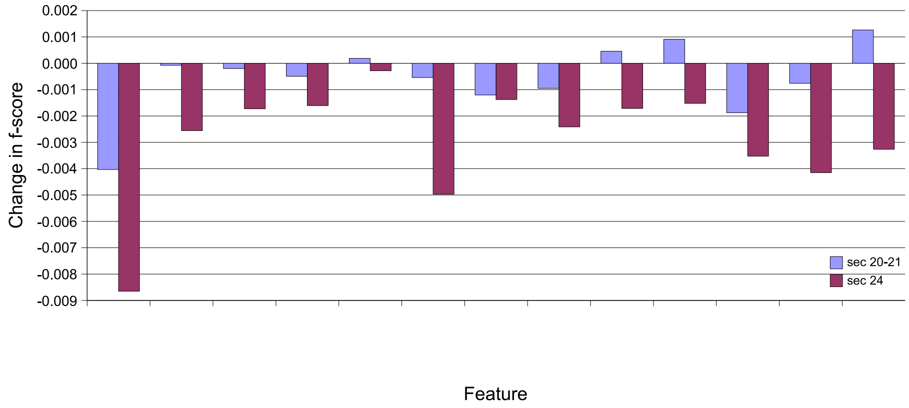
- The feature weights are not that indicative of how important a feature is
- The MaxEnt ranker with regularizer tuning takes approx 1 day to train
- The *averaged perceptron* algorithm takes approximately 2 minutes
  - used in experiments comparing different sets of features
  - Used to compare models with the following features: **NLogP Rule NGram Word WProj RightBranch Heavy NGramTree HeadTree Heads Neighbours CoPar CoLenPar**

Adding one feature class



- Averaged perceptron baseline with only base parser log prob feature
  - section 20–21 f-score = 0.894913
  - section 24 f-score = 0.889901

Subtracting one feature class



- Averaged perceptron baseline with all features
  - section 20–21 f-score = 0.906806
  - section 24 f-score = 0.902782

Comparing estimators

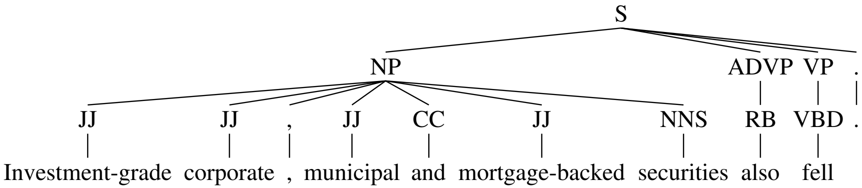
- Training on sections 2–19, regularizer tuned on 20–21, evaluate on 24

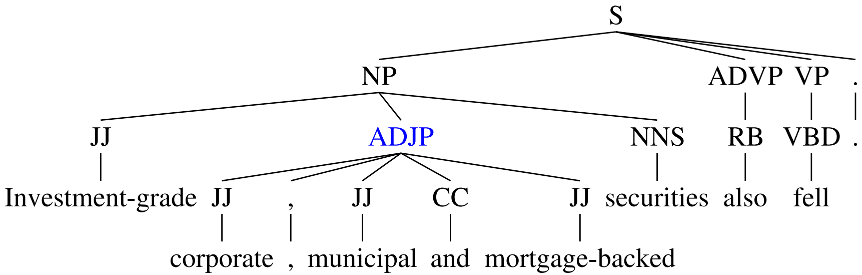
Estimator	# features	sec 20-21	sec 24
MaxEnt model, $p = 2$	670,688	0.9085	0.9037
MaxEnt model, $p = 1$	14,549	0.9078	0.9024
averaged perceptron	523,374	0.9068	0.9028
expected f-score	670,688	0.9084	0.9029

- None of the differences are significant
- Because the exponential model with $p = 2$ was the first model I tested new features on, they may be biased to work well with it.

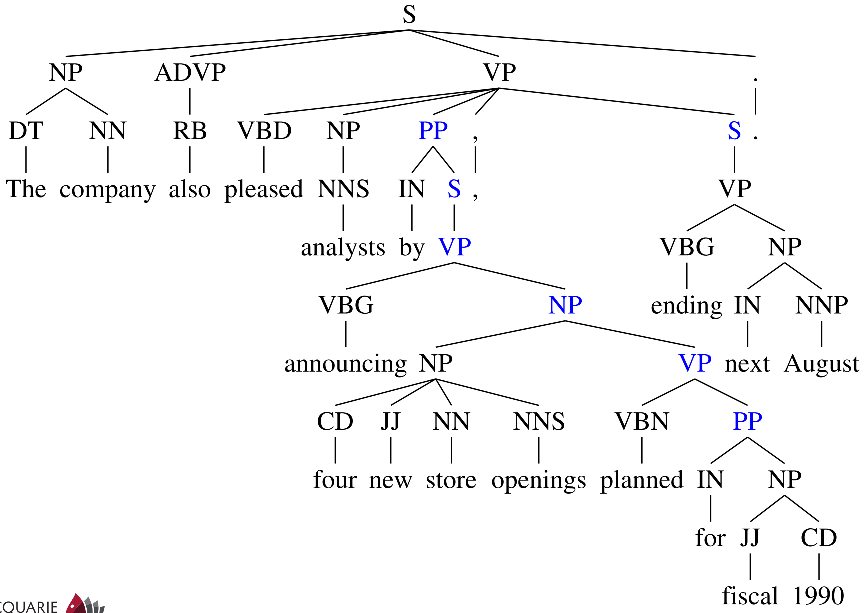
MACQUARIE UNIVERSITY

Sample parser output (1)

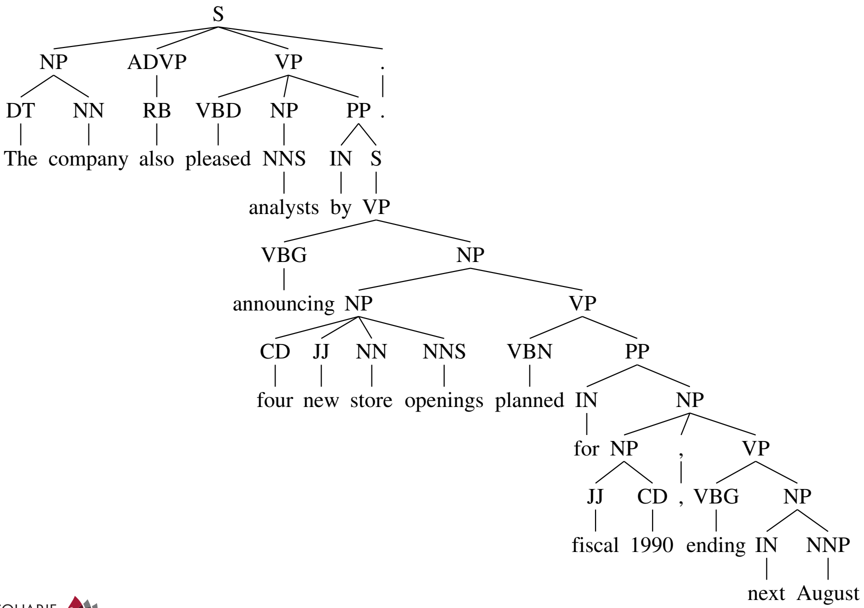




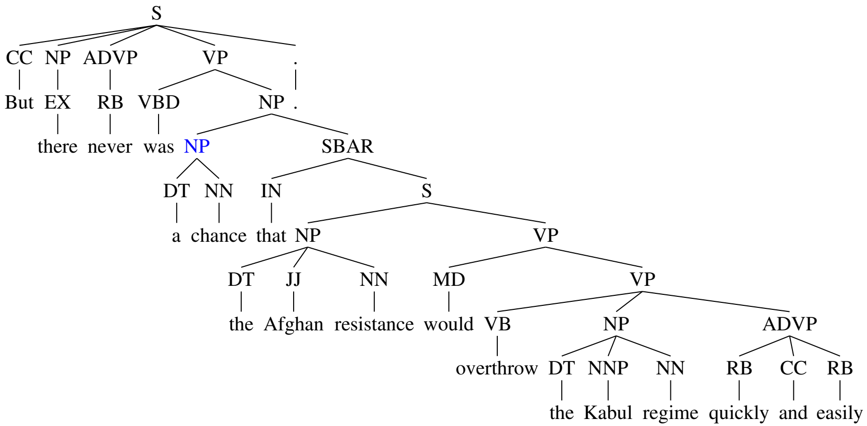
Sample parser output (2 parser)



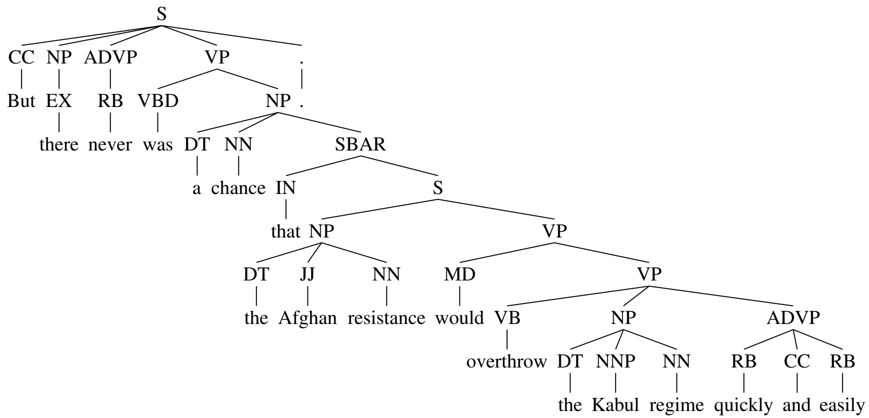
Sample parser output (2 gold)



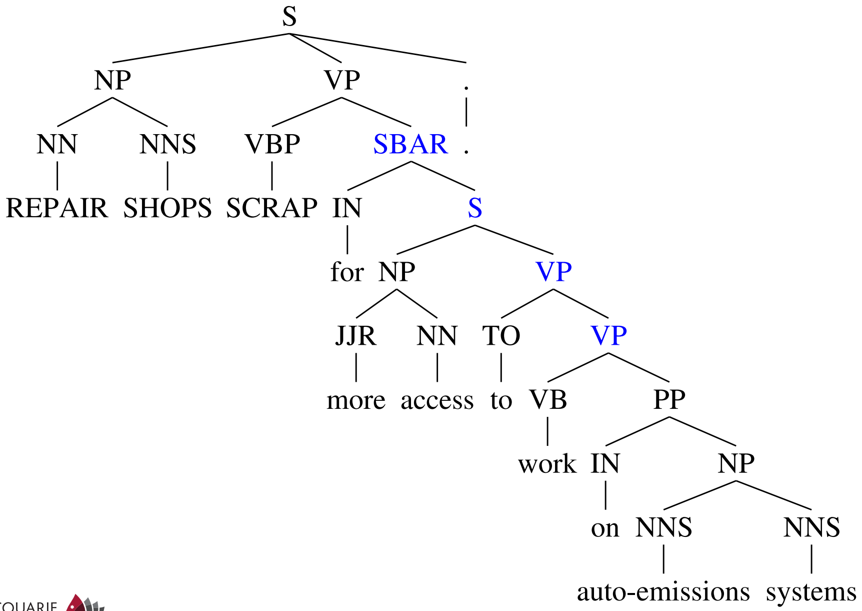
Sample parser output (3 parser)



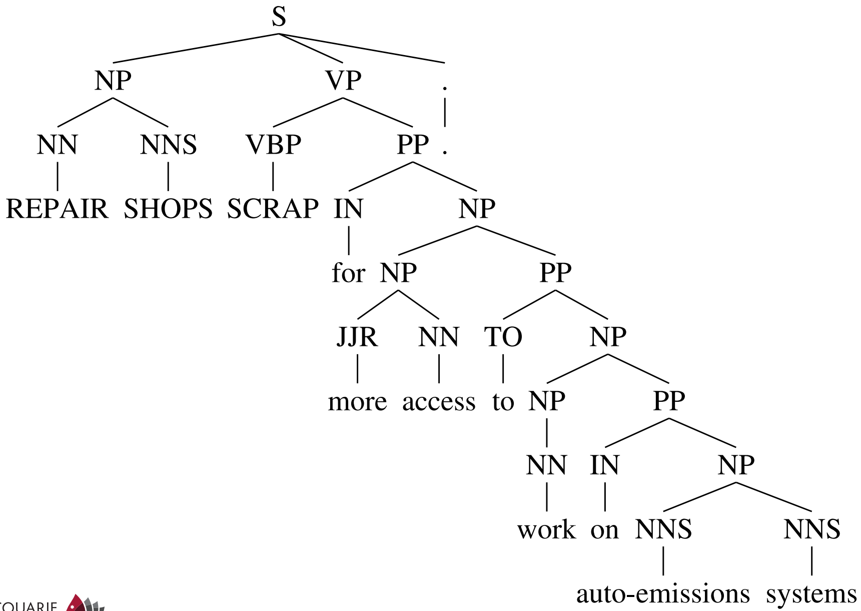
Sample parser output (3 gold)



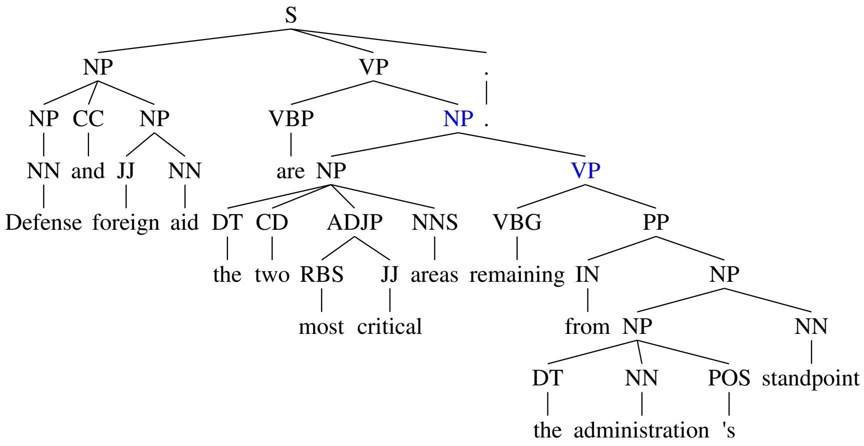
Sample parser output (4 parser)



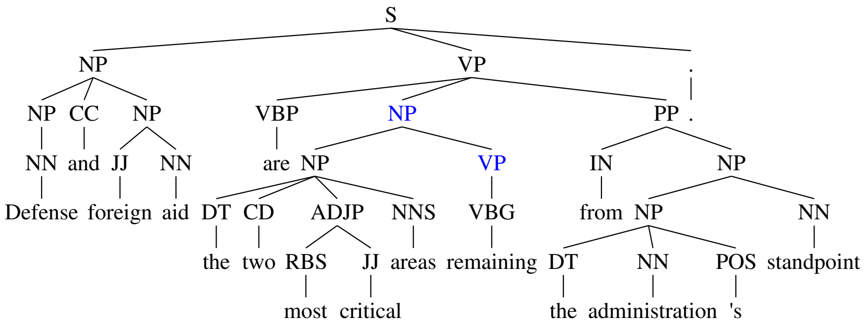
Sample parser output (4 gold)



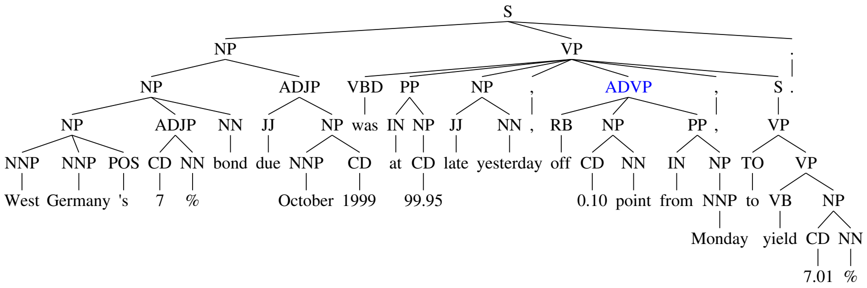
Sample parser output (5 parser)



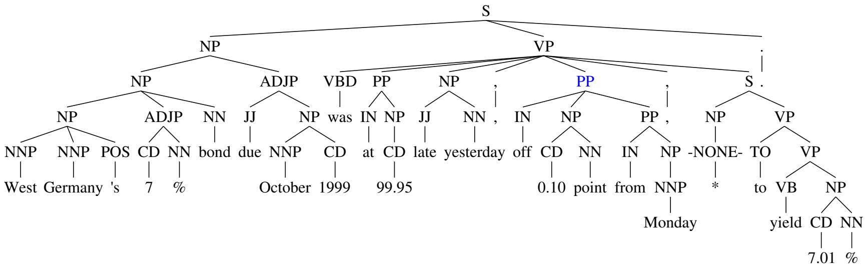
Sample parser output (5 gold)



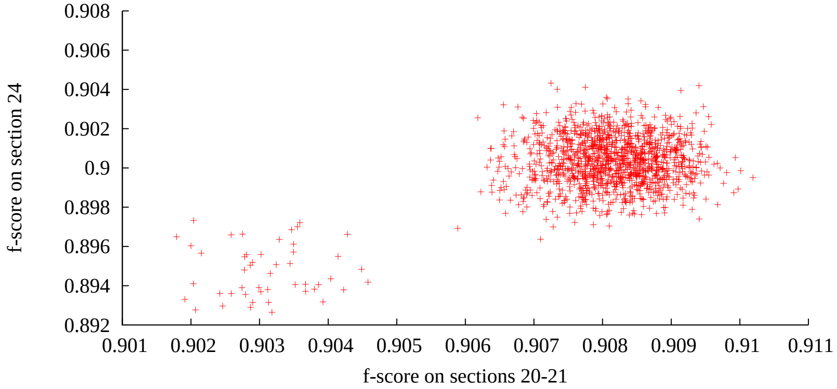
Sample parser output (6 parser)


S
NP VP .
NP ADJP VBD PP NP , ADVP , S .
NP ADJP NN JJ NP was IN NP JJ NN , RB NP PP , VP
NNP NNP POS CD NN bond due NNP CD at CD late yesterday off CD NN IN NP TO VP
West Germany 's 7 % October 1999 99.95 0.10 point from NNP to VB NP
Monday yield CD NN
7.01 %

Sample parser output (6 gold)



Variation in parser accuracy on different test sets



- Each point shows the *accuracy of a reranker trained using the averaged perceptron algorithm*

Further directions in syntactic parsing

- *Self-training:* semi-supervised learning using the parser's trees as input for training
- *Automatic domain adaptation:* selecting an optimal combination of treebanks for parser training based on properties of the corpus
- *Parsing speech recognizer output:* detecting and partially correcting disfluencies; using prosody in place of punctuation

Conclusion

- It's possible to define probability distributions over essentially any kind of linguistic objects

$\Rightarrow$ *No conflict between statistics and linguistics*

- If the model is *generative* (i.e., products of conditional distributions)
  - partition function $Z = 1$
  - $\Rightarrow$ ML estimation is easy (no need to sum over all possible structures)
  - $\Rightarrow$ EM algorithm for learning from hidden data
- If the model includes arbitrary "context-sensitive" dependencies
  - partition function $Z \neq 1$
  - $\Rightarrow$ ML estimation is intractible (need to sum over all possible structures)
  - Conditional ML estimation is possible, but computationally demanding
  - $\Rightarrow$ EM algorithm for learning from hidden data is intractible

Outline

Overview

n-gram language modelling

Machine translation

Sequence tagging with Hidden Markov Models

Grammars and Syntactic Parsing

PCFGs and beyond for statistical parsing

Unsupervised learning with non-parametric Bayesian models

Ideas behind talk

- Statistical methods have revolutionized computational linguistics and cognitive science
- But most successful learning methods are *parametric*
  - learn values of a *fixed number of parameters*
- *Non-parametric Bayesian methods* learn the parameters
- *Adaptor Grammars* learn probability of each *adapted subtree*
  - c.f., data-oriented parsing
- *"Rich get richer"* learning rule $\Rightarrow$ *Zipf distributions*
- Applications of Adaptor Grammars:
  - acquisition of *concatenative morphology*
  - *word segmentation* and lexical acquisition
  - learning the structure of *named-entity NPs*
- Sampling (instead of EM) is a natural approach to Adaptor Grammar inference

MACQUARIE UNIVERSITY

Language acquisition as Bayesian inference

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

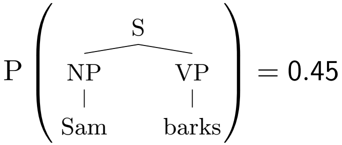
- Likelihood measures how well grammar describes data
- Prior expresses knowledge of grammar before data is seen
  - can be very specific (e.g., Universal Grammar)
  - can be very general (e.g., prefer shorter grammars)
- Posterior is a *distribution* over grammars
  - captures *learner's uncertainty* about which grammar is correct
- Language learning is *non-parametric* inference
  - no (obvious) bound on number of words, grammatical morphemes, etc

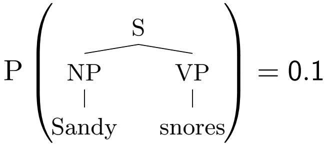
Outline

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
$\mathrm{S} \rightarrow \mathrm{NP\ VP}$	1.0		
$\mathrm{NP} \rightarrow \mathrm{Sam}$	0.75	$\mathrm{NP} \rightarrow \mathrm{Sandy}$	0.25
$\mathrm{VP} \rightarrow \mathrm{barks}$	0.6	$\mathrm{VP} \rightarrow \mathrm{snores}$	0.4



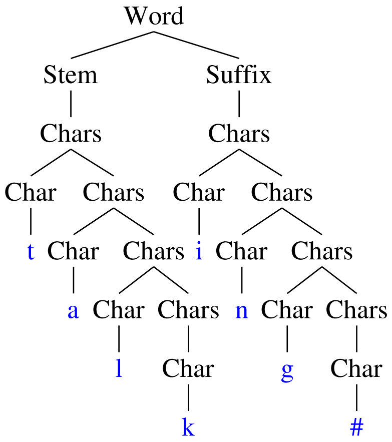


Learning syntactic structure is hard

- 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

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



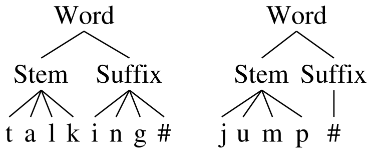
- Grammar's trees can represent any segmentation of words into stems and suffixes

⇒ 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

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



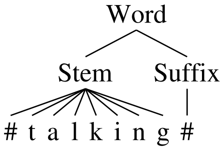
- A rule for each morpheme
  ⇒ "PCFG" can represent probability of each morpheme
- *Unbounded number of possible rules, so this is not a PCFG*
  - not a practical problem, as only a finite set of rules could possibly be used in any particular data set

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

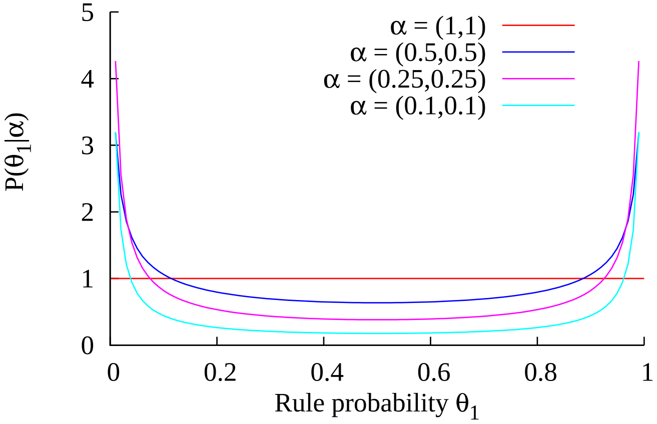
$\Rightarrow$ Saturated model is maximum likelihood estimate

- 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 $\theta_{A\to\beta} \approx 0$



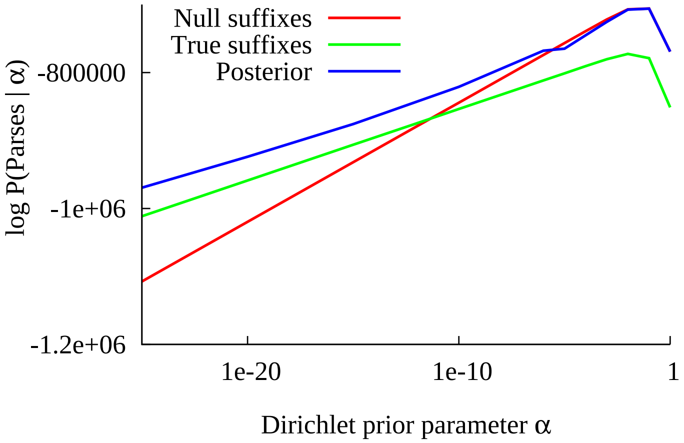
Morphological segmentation experiment

- 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	
reports		report	s	report	s	report	s
reported		reported		reported		reported	
reporting		report	ing	report	ing	report	ing

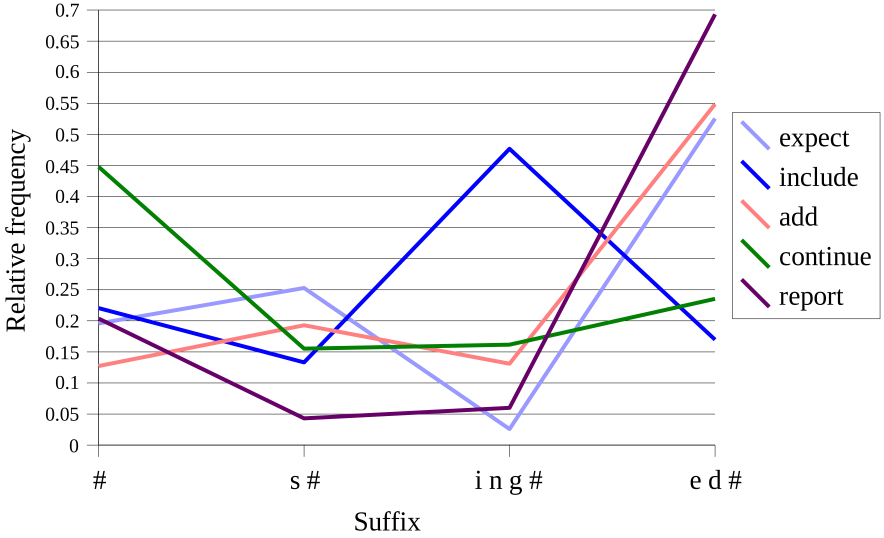
Log posterior for models on token data



- Correct solution is nowhere near as likely as posterior

$\Rightarrow$ model is wrong!

Relative frequencies of inflected verb forms



Types and tokens

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

Data	=	"the cat chased the other cat"
Tokens	=	"the", "cat", "chased", "the", "other", "cat"
Types	=	"the", "cat", "chased", "other"

- Estimating θ from *word types* rather than word tokens eliminates (most) frequency variation
  - 4 common verb suffixes, so when estimating from verb types $\theta_{\text{Suffix}\to\text{i n g \#}} \approx 0.25$
- Several psycholinguists believe that humans learn morphology from word types
- Adaptor grammar mimics Goldwater et al "Interpolating between Types and Tokens" morphology-learning model

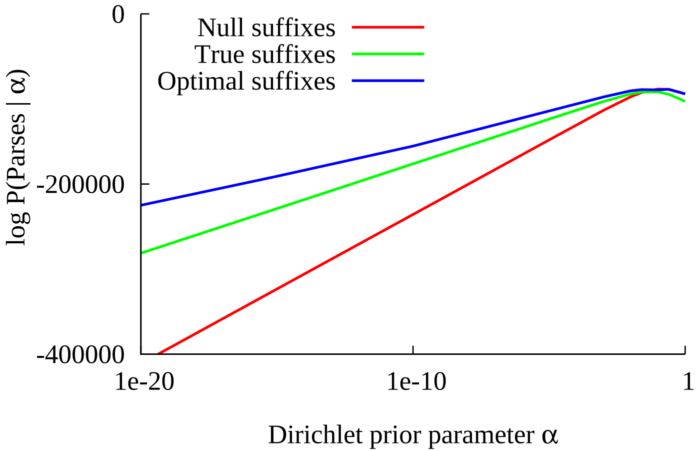
MACQUARIE UNIVERSITY

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
reports		report	s	repo	rts	rep	orts
reported		report	ed	repo	rted	rep	orted
report	ing	report	ing	repo	rting	rep	orting

MACQUARIE UNIVERSITY

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 naturally emerges*

Outline

MACQUARIE UNIVERSITY

Bayesian inference for Dirichlet-multinomials

- Probability of next event with *uniform Dirichlet prior* with mass α over m outcomes and observed data $\boldsymbol{Z}_{1:n} = (Z_1, \ldots, Z_n)$

$$\mathrm{P}(Z_{n+1} = k \mid \boldsymbol{Z}_{1:n}, \alpha) \quad \propto \quad n_k(\boldsymbol{Z}_{1:n}) + \alpha/m$$

  where $n_k(\boldsymbol{Z}_{1:n})$ is number of times k appears in $\boldsymbol{Z}_{1:n}$
- Example: Coin ($m = 2$), $\alpha = 1$, $\boldsymbol{Z}_{1:2} = (\mathrm{heads}, \mathrm{heads})$
  - $\mathrm{P}(Z_3 = \mathrm{heads} \mid \boldsymbol{Z}_{1:2}, \alpha) \propto 2.5$
  - $\mathrm{P}(Z_3 = \mathrm{tails} \mid \boldsymbol{Z}_{1:2}, \alpha) \propto 0.5$

Dirichlet-multinomials with many outcomes

- Predictive probability:

$$\mathrm{P}(Z_{n+1} = k \mid \boldsymbol{Z}_{1:n}, \alpha) \quad \propto \quad n_k(\boldsymbol{Z}_{1:n}) + \alpha/m$$

- Suppose the number of outcomes $m \gg n$. Then:

$$\mathrm{P}(Z_{n+1} = k \mid \boldsymbol{Z}_{1:n}, \alpha) \quad \propto \quad \begin{cases} n_k(\boldsymbol{Z}_{1:n}) & \text{if } n_k(\boldsymbol{Z}_{1:n}) > 0 \\ \alpha/m & \text{if } n_k(\boldsymbol{Z}_{1:n}) = 0 \end{cases}$$

- But *most outcomes will be unobserved*, so:

$$\mathrm{P}(Z_{n+1} \notin \boldsymbol{Z}_{1:n} \mid \boldsymbol{Z}_{1:n}, \alpha) \quad \propto \quad \alpha$$

MACQUARIE UNIVERSITY

From Dirichlet-multinomials to Chinese Restaurant Processes

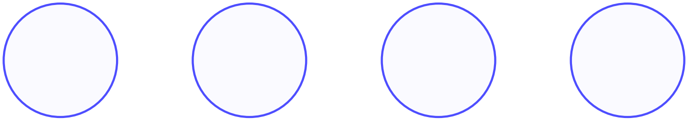
...

- Suppose *number of outcomes is unbounded* but *we* pick the event labels
- If we number event types in order of occurrence
  $\Rightarrow$ *Chinese Restaurant Process*

$$
\begin{aligned}
Z_1 &= 1 \\
\mathrm{P}(Z_{n+1} = k \mid \boldsymbol{Z}_{1:n}, \alpha) &\propto \begin{cases} n_k(\boldsymbol{Z}_{1:n}) & \text{if } k \leq m = \max(\boldsymbol{Z}_{1:n}) \\ \alpha & \text{if } k = m + 1 \end{cases}
\end{aligned}
$$

MACQUARIE UNIVERSITY

Chinese Restaurant Process (0)



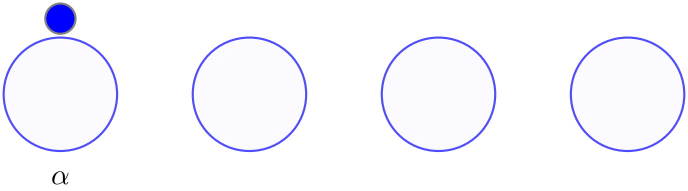
- Customer $\rightarrow$ table mapping $\boldsymbol{Z} =$
- $\mathrm{P}(\boldsymbol{z}) = 1$

- Next customer chooses a table according to:

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

MACQUARIE UNIVERSITY

Chinese Restaurant Process (1)

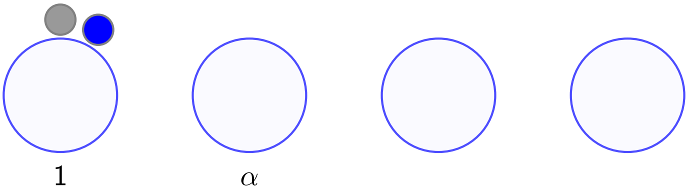


- Customer $\rightarrow$ table mapping $\boldsymbol{Z} = 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha$

- Next customer chooses a table according to:

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

Chinese Restaurant Process (2)

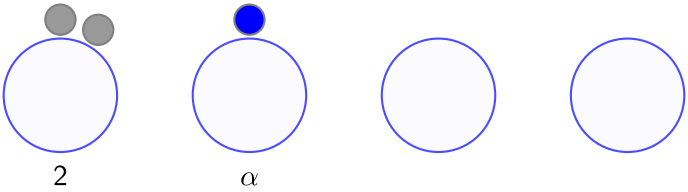


- Customer $\rightarrow$ table mapping $\boldsymbol{Z} = 1, 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha)$

- Next customer chooses a table according to:

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

Chinese Restaurant Process (3)

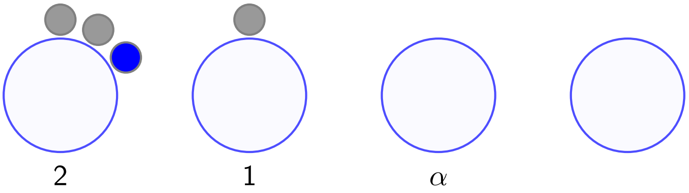


- Customer $\rightarrow$ table mapping $\boldsymbol{Z} = 1, 1, 2$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha) \times \alpha/(2+\alpha)$

- Next customer chooses a table according to:

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

Chinese Restaurant Process (4)

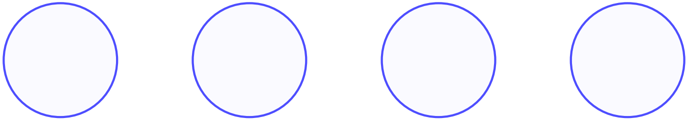


- Customer $\rightarrow$ table mapping $\boldsymbol{Z} = 1, 1, 2, 1$
- $\mathrm{P}(\boldsymbol{z}) = \alpha/\alpha \times 1/(1+\alpha) \times \alpha/(2+\alpha) \times 2/(3+\alpha)$

- Next customer chooses a table according to:

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

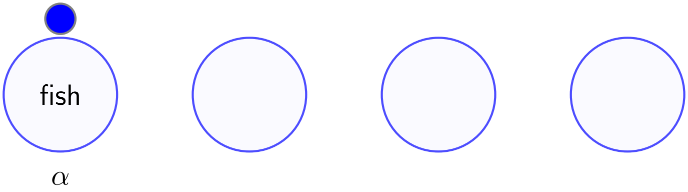
Labeled Chinese Restaurant Process (0)



- 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 $X_i = Y_{Z_i}$

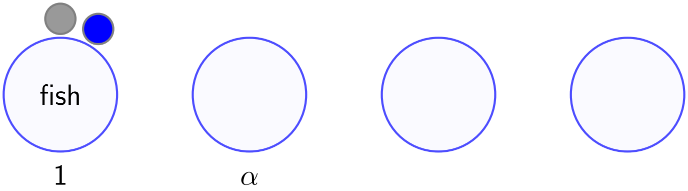
Labeled Chinese Restaurant Process (1)



- 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)

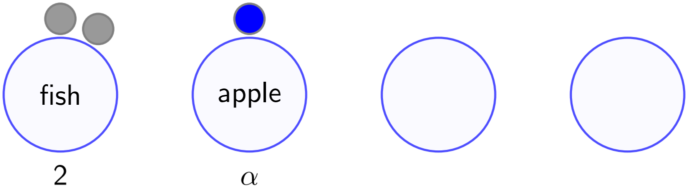


- Table $\rightarrow$ label mapping $\boldsymbol{Y}$ = fish
- Customer $\rightarrow$ table mapping $\boldsymbol{Z} = 1, 1$
- Output sequence $\boldsymbol{X}$ = fish,fish
- $\mathrm{P}(\boldsymbol{X}) = \mathrm{P}_0(\text{fish}) \times 1/(1+\alpha)$

- *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 $X_i = Y_{Z_i}$

MACQUARIE UNIVERSITY

Labeled Chinese Restaurant Process (3)

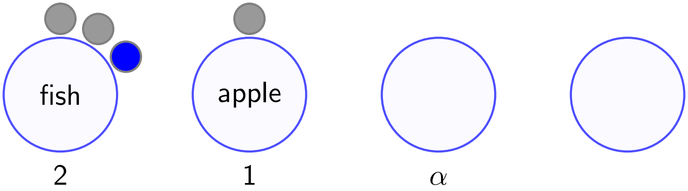


- Table $\rightarrow$ label mapping $\boldsymbol{Y}$ = fish,apple
- Customer $\rightarrow$ table mapping $\boldsymbol{Z} = 1, 1, 2$
- Output sequence $\boldsymbol{X}$ = fish,fish,apple
- $\mathrm{P}(\boldsymbol{X}) = \mathrm{P}_0(\text{fish}) \times 1/(1+\alpha) \times \alpha/(2+\alpha)\mathrm{P}_0(\text{apple})$

- *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 $X_i = Y_{Z_i}$

MACQUARIE UNIVERSITY

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 . . .

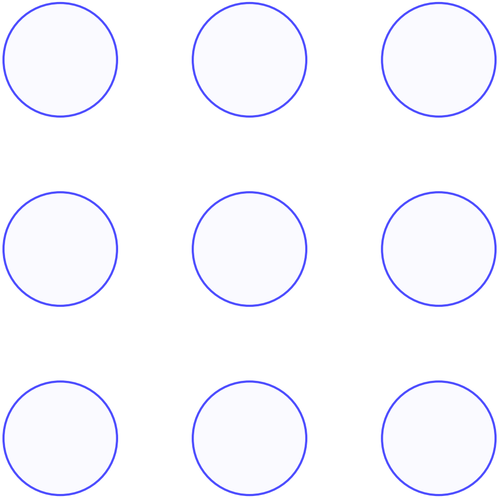
Outline

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

MACQUARIE UNIVERSITY

Adaptor grammar for stem-suffix morphology (0)



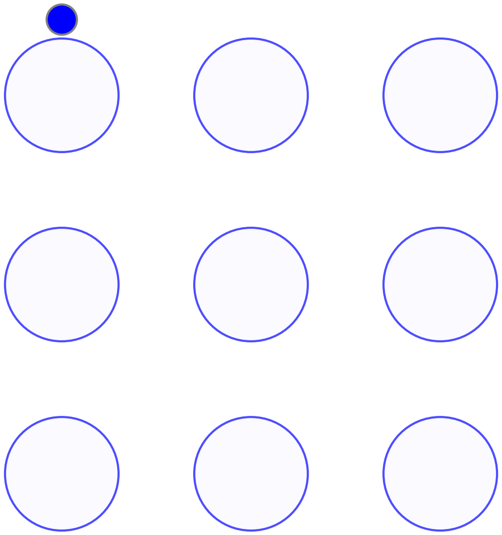
Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*

Generated words:

Adaptor grammar for stem-suffix morphology (1a)



Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*

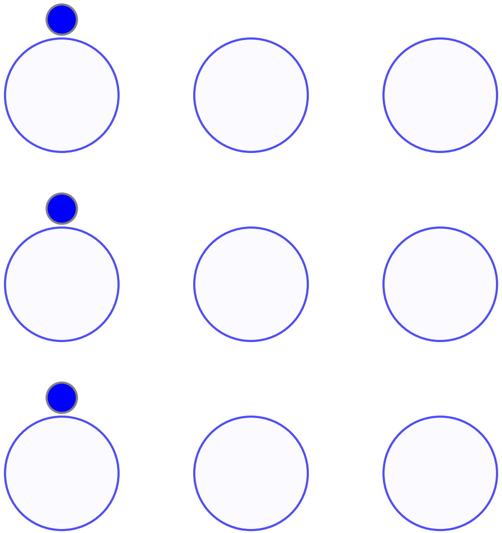
Generated words:

Adaptor grammar for stem-suffix morphology (1b)

$\underline{\text{Word}} \to \text{Stem Suffix}$

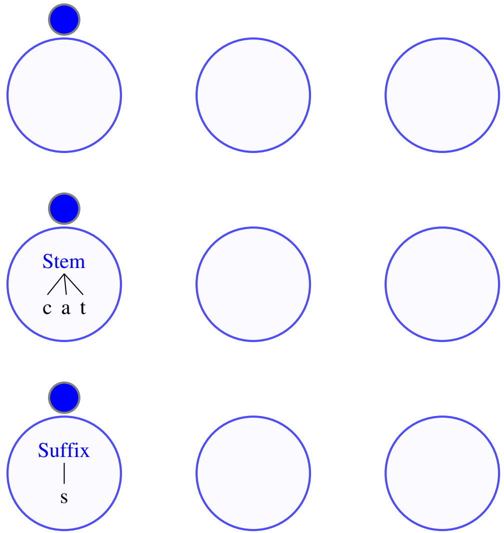
$\underline{\text{Stem}} \to \text{Phoneme}^{+}$

$\underline{\text{Suffix}} \to \text{Phoneme}^{\star}$



Generated words:

Adaptor grammar for stem-suffix morphology (1c)



$\underline{\text{Word}} \rightarrow \text{Stem Suffix}$

$\underline{\text{Stem}} \rightarrow \text{Phoneme}^{+}$

$\underline{\text{Suffix}} \rightarrow \text{Phoneme}^{\star}$

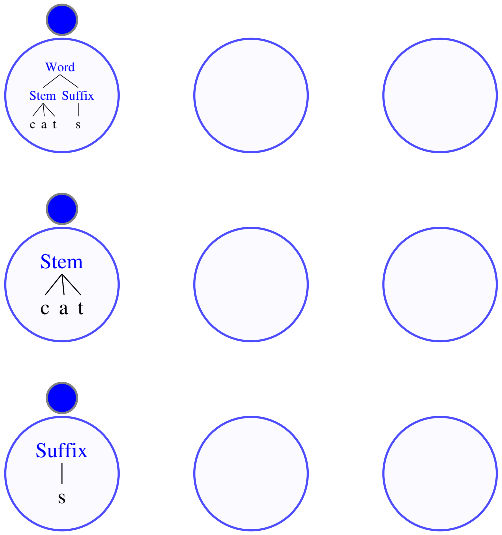
Generated words:

Adaptor grammar for stem-suffix morphology (1d)

$\underline{\text{Word}} \rightarrow \text{Stem Suffix}$

$\underline{\text{Stem}} \rightarrow \text{Phoneme}^{+}$

$\underline{\text{Suffix}} \rightarrow \text{Phoneme}^{\star}$



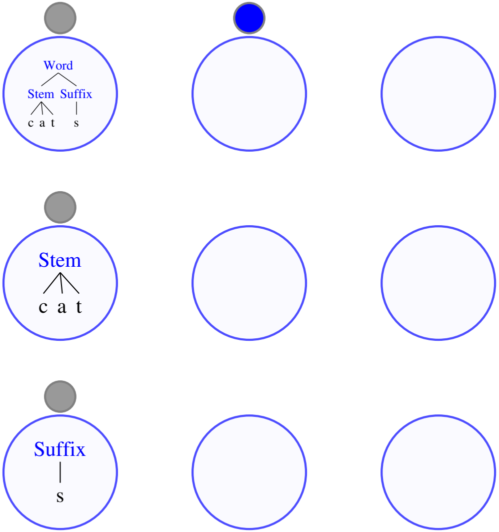
Generated words: cats

Adaptor grammar for stem-suffix morphology (2a)

Word → Stem Suffix

Stem → Phoneme^{+}

Suffix → Phoneme*



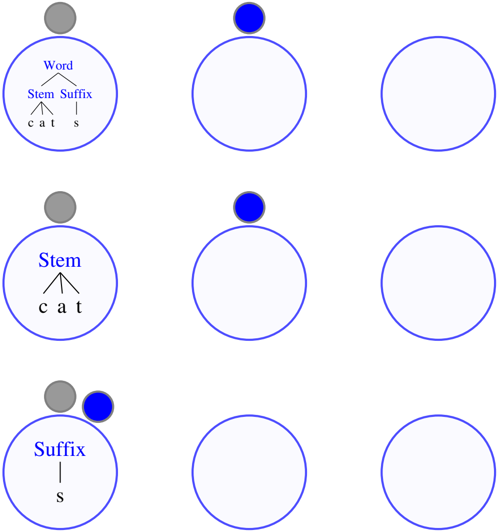
Generated words: cats

Adaptor grammar for stem-suffix morphology (2b)

Word $\rightarrow$ Stem Suffix

Stem $\rightarrow$ Phoneme^{+}

Suffix $\rightarrow$ Phoneme*



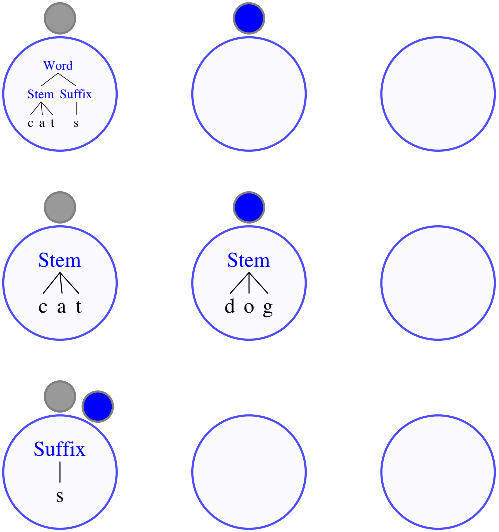
Generated words: cats

Adaptor grammar for stem-suffix morphology (2c)

<u>Word</u> → Stem Suffix

<u>Stem</u> → Phoneme$^+$

<u>Suffix</u> → Phoneme*



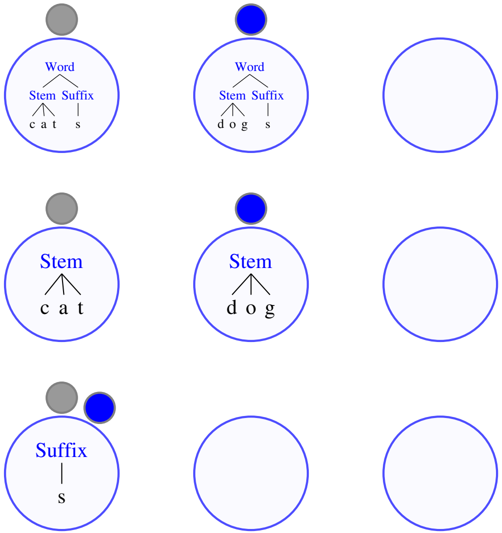
Generated words: cats

Adaptor grammar for stem-suffix morphology (2d)

Word → Stem Suffix

Stem → Phoneme$^+$

Suffix → Phoneme*



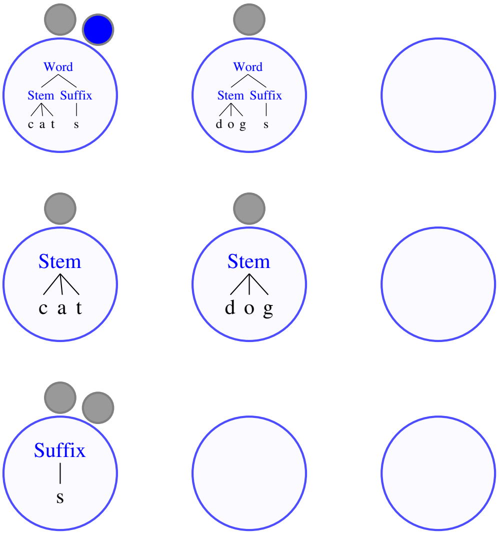
Generated words: cats, dogs

Adaptor grammar for stem-suffix morphology (3)

<u>Word</u> → Stem Suffix

<u>Stem</u> → Phoneme$^+$

<u>Suffix</u> → 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 t rooted in A with probability proportional to the number of times t was previously generated
  - using $A \to \beta$ with probability proportional to $\alpha_A \theta_{A\to\beta}$

MACQUARIE UNIVERSITY

Properties of adaptor grammars

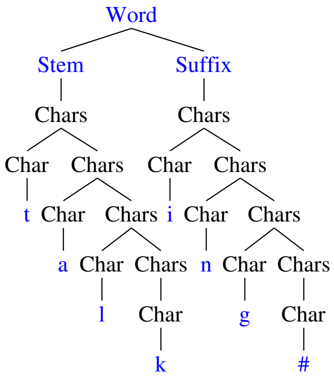
- Possible trees are generated by CFG rules
  but the probability of each adapted tree is learned separately
- Probability of adapted subtree t is proportional to:
  - the number of times t was seen before
    $\Rightarrow$ "rich get richer" dynamics (Zipf distributions)
  - plus α_A times prob. of generating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

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

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

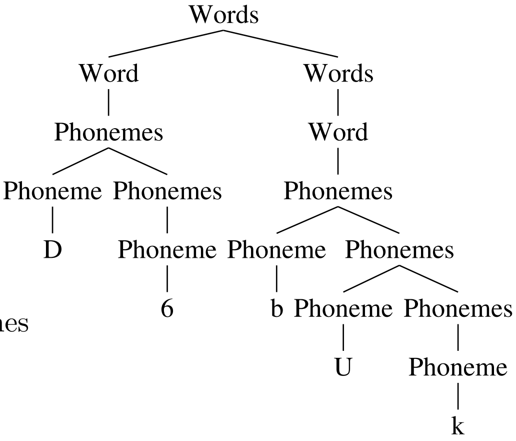
MACQUARIE UNIVERSITY

Word segmentation with PCFGs (1)

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

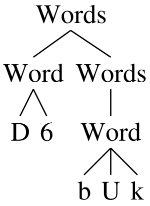
which abbreviates

$$\begin{aligned}
&\text{Sentence} \to \text{Words}\\
&\text{Words} \to \text{Word Words}\\
&\text{Word} \to \text{Phonemes}\\
&\text{Phonemes} \to \text{Phoneme Phonemes}\\
&\text{Phonemes} \to \text{Phoneme}\\
&\text{Phoneme} \to a \mid \ldots \mid z
\end{aligned}$$



Word segmentation with PCFGs (1)

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

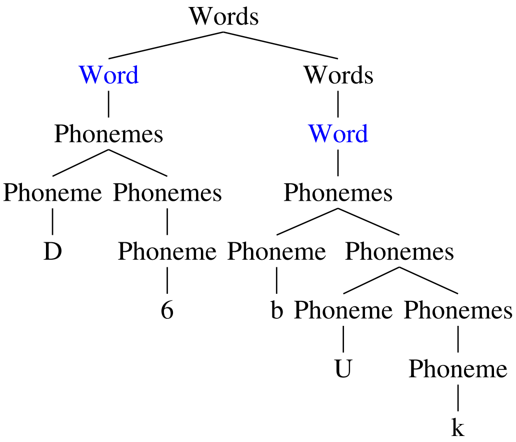


- But now there are an infinite number of PCFG rules!
  - once we see our (finite) training data, only finitely many are useful
  - $\Rightarrow$ the set of parameters (rules) should be chosen based on training data

Unigram word segmentation adaptor grammar

Sentence → Word$^+$
Word → Phoneme$^+$

- *Adapted nonterminals* depicted in blue



- 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

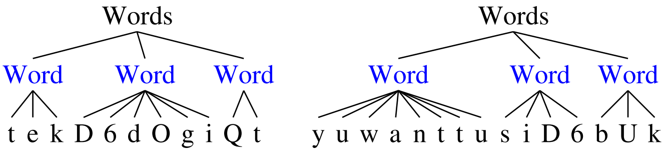
- *A grammar learnt from Brent corpus*

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)

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

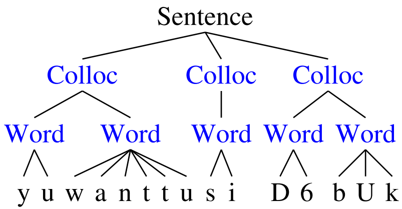
- Unigram word segmentation model assumes each word is generated independently
- But there are strong inter-word dependencies (collocations)
- Unigram model can only capture such dependencies by analyzing collocations as words (Goldwater 2006)



- Note: *unadapted nodes suppressed in this and later trees*

Collocations ⇒ Words

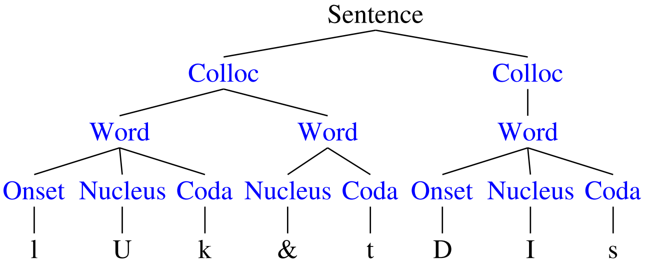
$$\begin{aligned}\text{Sentence} &\to \text{Colloc}^{+}\\ \underline{\text{Colloc}} &\to \text{Word}^{+}\\ \underline{\text{Word}} &\to \text{Phon}^{+}\end{aligned}$$



- 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$^+$

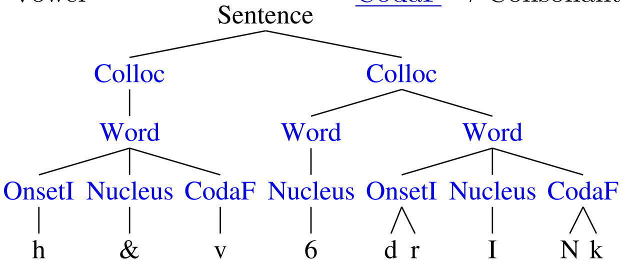


- With no supra-word generalizations, f-score = 68%
- With 2 Collocation levels, f-score = 82%

Distinguishing internal onsets/codas helps

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

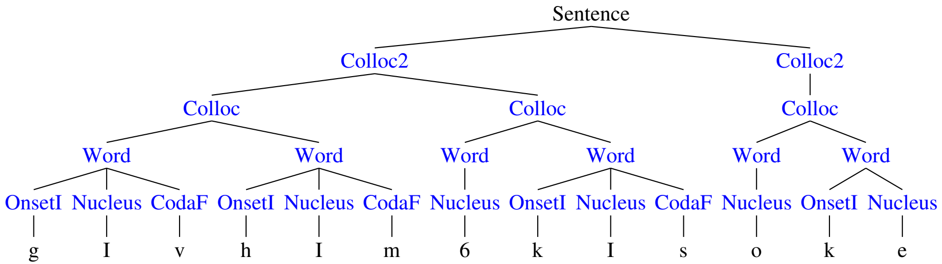
Colloc → Word$^+$
Word → SyllableI SyllableF
SyllableIF → (OnsetI) RhymeF
RhymeF → Nucleus (CodaF)
CodaF → Consonant$^+$



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

MACQUARIE UNIVERSITY

Collocations^2 ⇒ Words ⇒ Syllables



Summary: Adaptor grammars for word segmentation

- Word segmentation accuracy depends on the kinds of generalisations learnt.

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

- *Accuracy improves when you learn more*
  - *explain away* potentially misleading generalizations

Outline

MACQUARIE
UNIVERSITY

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
- If we aren't given the parse trees:
  - there can be *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

Monte Carlo sampling approaches to inference

- Suppose we'd like to know $\mathrm{P}(y)$, but it's too hard to compute
- Idea: produce *samples* $y^{(1)}, y^{(2)}, \ldots$ from $\mathrm{P}(y)$
  - there are generic methods for producing samples
- Given samples $y^{(1)}, \ldots, y^{(n)}$ from $\mathrm{P}(y)$, we can approximate the *expected value* of any function f of y by:

$$\mathrm{E}[f] \;=\; \sum_{y \in \mathcal{Y}} f(y)\,\mathrm{P}(y) \;\approx\; \frac{1}{n} \sum_{j=1}^{n} f(y)$$

- For word segmentation adaptor grammars
  - produce *sample parses* of each sentence
  - compute expected value of a word boundary at each possible boundary point

Gibbs sampling

- Gibbs sampling is a generic procedure for producing samples from *multi-dimensional variables*
- Goal: sample from $\mathrm{P}(y_1, \ldots, y_n)$
- generic Gibbs sampler algorithm:
    - initialise $\boldsymbol{y} = (y_1, \ldots, y_n)$ somehow
    - repeat forever:
        - pick an index $j \in 1, \ldots, n$ at random
        - replace y_j with a sample from $\mathrm{P}(y_j \mid y_{-j})$, where $\boldsymbol{y}_{-j} = (y_1, \ldots, y_{j-1}, y_{j+1}, \ldots, y_n)$
- For a wide range of distributions, these samples converge on $\mathrm{P}(\boldsymbol{y})$

A Gibbs sampler for adaptor grammars

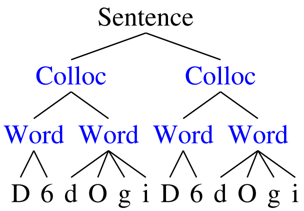
- We'd like to sample $\mathrm{P}(\boldsymbol{t} \mid \boldsymbol{w})$, where:
  - $\boldsymbol{w} = (w_1, \ldots, w_n)$, where each w_j is a sequence of terminals
  - $\boldsymbol{t} = (t_1, \ldots, t_n)$, where t_j is a parse for w_j
- Output: a sequence of sample parse trees $\boldsymbol{t}^{(1)}, \boldsymbol{t}^{(2)}, \ldots$ where each $\boldsymbol{t}^{(u)} = (t_1^{(u)}, \ldots, t_n^{(u)})$ contains a parse $t_j^{(u)}$ for w_j
- 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 $A \rightarrow \textsc{Yield}(t)$ for each table t in A's restaurant: $\theta'_{A\rightarrow\textsc{Yield}(t)} \propto n_t$, the number of customers at table t
- Map parses using $G'(\boldsymbol{t}_{-j})$ 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

MACQUARIE UNIVERSITY

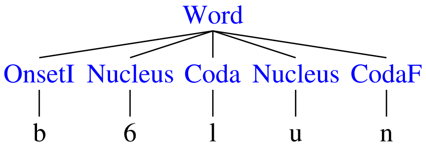
Outline

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 ⇒ learns wrong grammar half the time

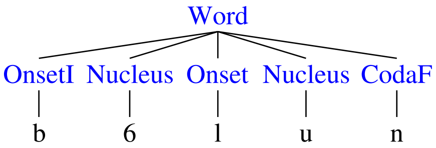


- Syllabification accuracy is relatively poor
  Syllabification given true word boundaries: f-score = 83%
  Syllabification learning word boundaries: f-score = 74%

Preferring Onsets improves syllabification

2 Syllable $\rightarrow$ Onset Rhyme   1 Syllable $\rightarrow$ Rhyme

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



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

Modelling sonority classes improves syllabification

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

- 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%

Outline

The Sesotho corpus

- Sesotho is a Bantu language spoken in southern Africa
- Orthography is (roughly) phonemic
  ⇒ use orthographic forms as broad phonemic representations
- Rich agglutinative morphology (especially in verbs)

u-	e-	nk-	il-	e	kae
SM-	OM-	take-	PERF-	IN	where

  "You took it from where?"

- The Demuth Sesotho corpus (1992) contains transcripts of child and child-directed speech
- Here used a subset of size roughly comparable to Brent corpus of infant-directed speech

	Brent	Demuth
utterances	9,790	8,503
word tokens	33,399	30,200
phonemes	95,809	100,113

MACQUARIE UNIVERSITY

Sesotho verbs are morphologically complex

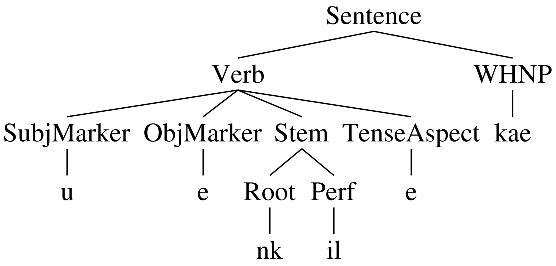
u-	e-	nk-	il-	e	kae
SM-	OM-	take-	PERF-	IN	where

"You took it from where?"

- Input:

  u ▵ e ▵ n ▵ k ▵ i ▵ l ▵ e ▵ k ▵ a ▵ e

- What I'd like to be able to learn someday:

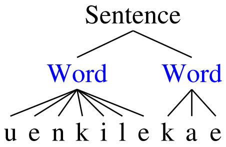


Unigram segmentation grammar – word

u- e- nk- il- e	kae
SM-OM-take-PERF-IN	where

“You took it from where?”

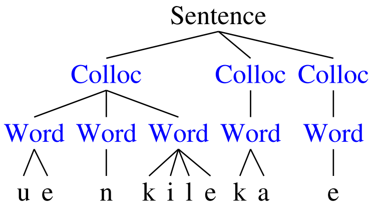
$$\begin{aligned}\text{Sentence} &\to \text{Word}^{+}\\ \underline{\text{Word}} &\to \text{Phoneme}^{+}\end{aligned}$$



- The word grammar has a word segmentation f-score of 43%
- Lower than 56% f-score on the Brent corpus.
- Sesotho words are longer and more complex.

Collocation grammar – colloc

Sentence → Colloc$^+$
Colloc → Word$^+$
Word → Phoneme$^+$



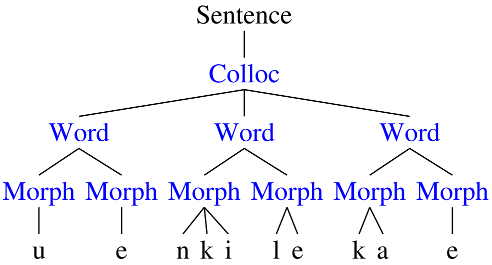
- Goldwater et al (2006) found that modelling bigram dependencies greatly improved English segmentation accuracy
- Johnson (2008) showed similar improvements by learning English collocations
- If we treat lower-level units as Words, f-score = 27%
- If we treat upper-level units as Words, f-score = 48%
- English improves by learning dependencies above words, but Sesotho improves by learning generalizations below words

Adding more levels – colloc2

u-	e-	nk-	il-	e	kae
SM-	OM-	take-	PERF-	IN	where

"You took it from where?"

Sentence → Colloc$^+$
Colloc → Word$^+$
Word → Morph$^+$
Morph → Phoneme$^+$



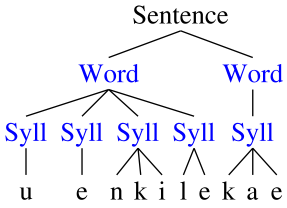
- If two levels are good, maybe three would be better?
- Word segmentation f-score drops to 47%
- Doesn't seem to be much value in adding dependencies above Word level in Sesotho

Using syllable structure – word – syll

u- e- nk- il- e	kae
SM-OM-take-PERF-IN	where

"You took it from where?"

$$
\begin{aligned}
&\text{Sentence} \to \text{Word}^{+}\\
&\underline{\text{Word}} \to \text{Syll}^{+}\\
&\underline{\text{Syll}} \to (\text{Onset})\,\text{Nuc}\,(\text{Coda})\\
&\underline{\text{Syll}} \to \text{SC}\\
&\text{Onset} \to \text{C}^{+}\\
&\text{Nuc} \to \text{V}^{+}\\
&\text{Coda} \to \text{C}^{+}
\end{aligned}
$$



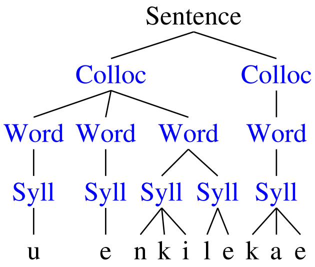
- SC (syllabic consonants) are '*l*', '*m*' '*n*' and '*r*'
- Word segmentation f-score = 50%
- Assuming that words are composed of syllables does improve Sesotho word segmentation

Using syllable structure – colloc – syll

u-	e-	nk-	il-	e	kae
SM-	OM-	take-	PERF-	IN	where

"You took it from where?"

$$
\begin{aligned}
&\text{Sentence} \rightarrow \text{Colloc}^{+}\\
&\underline{\text{Colloc}} \rightarrow \text{Word}^{+}\\
&\underline{\text{Syll}} \rightarrow (\text{Onset})\,\text{Nuc}\,(\text{Coda})\\
&\underline{\text{Syll}} \rightarrow \text{SC}\\
&\text{Onset} \rightarrow \text{C}^{+}\\
&\text{Nuc} \rightarrow \text{V}^{+}\\
&\text{Coda} \rightarrow \text{C}^{+}
\end{aligned}
$$



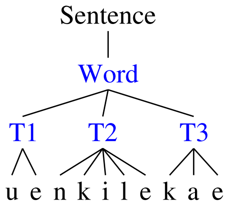
- Word segmentation f-score = 48%
- Additional collocation level doesn't help

Morpheme positions – word — morph

u-	e-	nk-	il-	e	kae
SM-	OM-	take-	PERF-	IN	where

"You took it from where?"

$$\text{Sentence} \to \text{Word}^+$$
$$\underline{\text{Word}} \to \text{T1}\,(\text{T2}\,(\text{T3}\,(\text{T4}\,(\text{T5}))))$$
$$\underline{\text{T1}} \to \text{Phoneme}^+$$
$$\underline{\text{T2}} \to \text{Phoneme}^+$$
$$\underline{\text{T3}} \to \text{Phoneme}^+$$
$$\underline{\text{T4}} \to \text{Phoneme}^+$$
$$\underline{\text{T5}} \to \text{Phoneme}^+$$



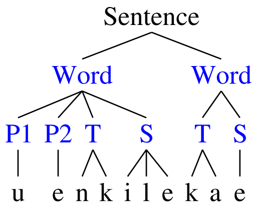
- Each word consists of 1–5 morphemes
- Learn separate morphemes for each position
- Improves word segmentation f-score to 53%

Building in language-specific information – word – smorph

u-	e-	nk-	il-	e	kae
SM-	OM-	take-	PERF-	IN	where

"You took it from where?"

$$
\begin{aligned}
&\text{Sentence} \rightarrow \text{Word}^{+} \\
&\underline{\text{Word}} \rightarrow (\text{P1}\,(\text{P2}\,(\text{P3})))\,\text{T}\,(\text{S}) \\
&\underline{\text{P1}} \rightarrow \text{Phoneme}^{+} \\
&\underline{\text{P2}} \rightarrow \text{Phoneme}^{+} \\
&\underline{\text{P3}} \rightarrow \text{Phoneme}^{+} \\
&\underline{\text{T}} \rightarrow \text{Phoneme}^{+} \\
&\underline{\text{S}} \rightarrow \text{Phoneme}^{+}
\end{aligned}
$$



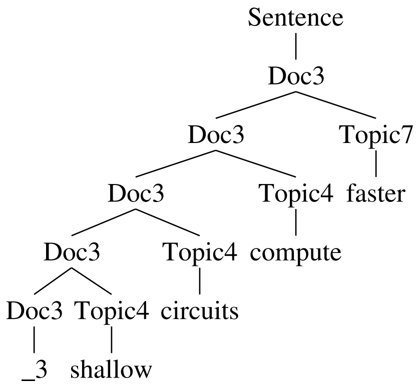
- In Sesotho many words consist of a stem T, an optional suffix S and up to 3 prefixes P1, P2 and P3
- Achieves highest f-score = 56%

Outline

MACQUARIE UNIVERSITY

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*

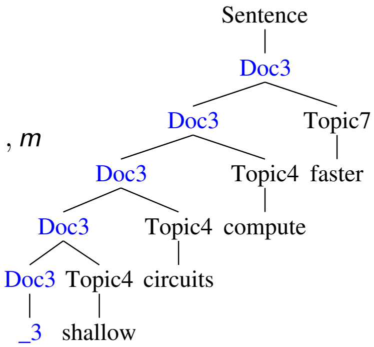


$$\begin{array}{lll} \text{Sentence} \to & \text{Doc}_i & i \in 1, \ldots, n \\ \text{Doc}_i \to & \_i & i \in 1, \ldots, n \\ \text{Doc}_i \to & \text{Doc}_i\ \text{Topic}_j & i \in 1, \ldots, n; j \in 1, \ldots, m \\ \text{Topic}_j \to & x & j \in 1, \ldots, m; x \in \mathcal{X} \end{array}$$

Left spine identifies document

- Left spine passes document id throughout sentence

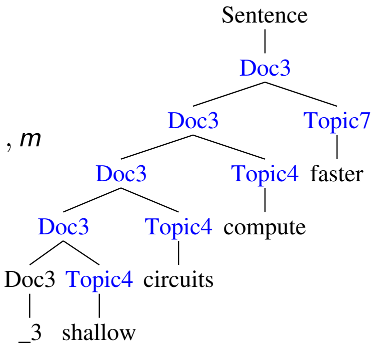
$\text{Sentence} \to$	Doc_i	$i \in 1, \ldots, n$
$\text{Doc}_i \to$	$\_i$	$i \in 1, \ldots, n$
$\text{Doc}_i \to$	$\text{Doc}_i\ \text{Topic}_j$	$i \in 1, \ldots, n; j \in 1, \ldots, m$
$\text{Topic}_j \to$	x	$j \in 1, \ldots, m; x \in \mathcal{X}$



Document → topic rules

- Document → topic rules specify probability of topic within document

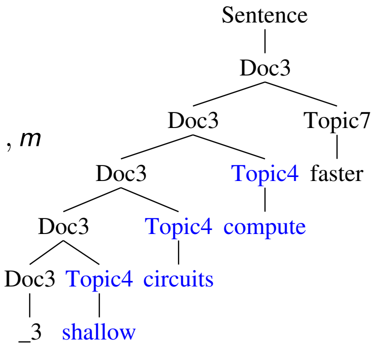
$$
\begin{array}{lll}
\text{Sentence} \rightarrow & \text{Doc}_i & i \in 1, \ldots, n \\
\text{Doc}_i \rightarrow & \_i & i \in 1, \ldots, n \\
\text{Doc}_i \rightarrow & \text{Doc}_i\ \text{Topic}_j & i \in 1, \ldots, n; j \in 1, \ldots, m \\
\text{Topic}_j \rightarrow & x & j \in 1, \ldots, m; x \in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



Topic → word rules

- Topic → word rules specify probability of word within topic

Sentence →	Doc_i	$i \in 1, \ldots, n$
Doc_i →	$\_i$	$i \in 1, \ldots, n$
Doc_i →	$\mathrm{Doc}_i\ \mathrm{Topic}_j$	$i \in 1, \ldots, n; j \in 1, \ldots, m$
Topic_j →	x	$j \in 1, \ldots, m; x \in \mathcal{X}$



Bayesian inference for LDA PCFGs

- Dirichlet priors on Document $\rightarrow$ Topic and Topic $\rightarrow$ Word distributions
- General-purpose PCFG parsing/estimation algorithms require time cubic in length of sentence
  - not a good idea for long documents!
- More efficient algorithms for these kinds of grammars
  - (standard LDA inference algorithms)
  - predictive (e.g., Earley) parsing algorithms
  - identify compositions finite state automata/transducers that these grammars encode

Outline

Learning words and their referents

PIG|DOG $\mathrm{I} \vartriangle \mathrm{z} \blacktriangle \mathrm{D} \vartriangle \& \vartriangle \mathrm{t} \blacktriangle \mathrm{D} \vartriangle 6 \blacktriangle \underbrace{\mathrm{p} \vartriangle \mathrm{I} \vartriangle \mathrm{g}}_{\text{PIG}}$

- Input to learner:
  - unsegmented phoneme sequence, and
  - *objects in nonlinguistic context*
- Learning objectives:
  - segment utterances into words, and
  - *learn word-object relationship*

AG for (unigram) segmentation and reference

- Each sentence is associated with *a single referent* from the non-linguistic context or a "null referent"
- Each word is either generated by the sentence's *referent-specific distribution* or the *null referent distribution*
- Given *possible referents* $\mathcal{R}$, the grammar contains rules:

$$
\begin{array}{ll}
\text{Sentence} \rightarrow \text{Referent}_r & \text{for each } r \in \mathcal{R} \\
\text{Referent}_r \rightarrow s & \text{for each } r \in \mathcal{R},\ r \in s \in 2^{\mathcal{R}} \\
\text{Referent}_r \rightarrow \text{Referent}_r\ \text{Word}_r & \text{for each } r \in \mathcal{R} \\
\text{Referent}_r \rightarrow \text{Referent}_r\ \text{Word}_\emptyset & \text{for each } r \in \mathcal{R} \\
\underline{\text{Word}}_r \rightarrow \text{Phonemes}_r & \text{for each } r \in \mathcal{R} \cup \{\emptyset\} \\
\text{Phonemes}_r \rightarrow \text{Phoneme}^+ & \text{for each } r \in \mathcal{R} \cup \{\emptyset\}
\end{array}
$$

- Sample parses:
  - T_dog|pig (Word I z D & t) (Word_dog D 6 d O g i)
  - T_dog|pig (Word D E r z) (Word_dog D 6 d O g i)
  - T_dog|pig (Word D & t s 6) (Word_pig p I g)

Joint segmentation and reference results

- Referents have much stronger effect on collocation model than on unigram model
- Best results with collocation grammar requiring zero or one referential word per sentence
  - requiring that referential words are surrounded by non-referential words may be as good
- Simultaneously learning word segmentation and reference *does not* improve average word segmentation (Tcolloc1 72% with real labels, 70% with permuted labels)
  - non-linguistic context is very impoverished
  - relatively few words are referential
- *Referential words are segmented better when referents are provided* (Tcolloc1 74% with real labels, 52% with permuted labels)

Outline

MACQUARIE UNIVERSITY

Adaptor grammars for 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

$\mathrm{NP} \rightarrow \mathrm{Unordered}^{+}$	$\underline{\mathrm{Unordered}} \rightarrow \mathrm{Word}^{+}$
$\mathrm{NP} \rightarrow (\mathrm{A0})\ (\mathrm{A1})\ \ldots\ (\mathrm{A6})$	$\mathrm{NP} \rightarrow (\mathrm{B0})\ (\mathrm{B1})\ \ldots\ (\mathrm{B6})$
$\underline{\mathrm{A0}} \rightarrow \mathrm{Word}^{+}$	$\underline{\mathrm{B0}} \rightarrow \mathrm{Word}^{+}$
$\ldots$	$\ldots$
$\underline{\mathrm{A6}} \rightarrow \mathrm{Word}^{+}$	$\underline{\mathrm{B6}} \rightarrow \mathrm{Word}^{+}$

- *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 can be *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

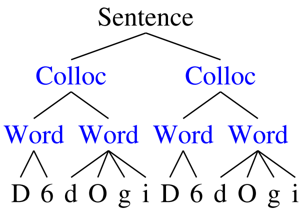
- Intuition: same as simple incremental algorithm, but re-use sentences in training data
  - Assign (random) parse trees to each sentence, and compute rule and subtree counts
  - Repeat forever:
    - pick a sentence (and corresponding parse) at random
    - deduct the counts for the sentence's parse 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 $A \to \beta$ from base PCFG: $\theta'_{A\to\beta} \;\propto\; \alpha_A \theta_{A\to\beta}$
  - A rule $A \to \mathrm{Y{\scriptsize IELD}}(t)$ for each table t in A's restaurant: $\theta'_{A\to\mathrm{Y{\scriptsize IELD}}(t)} \;\propto\; n_t$, the number of customers at table t
- Map parses using $G'(\boldsymbol{t}_{-j})$ 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

MACQUARIE UNIVERSITY

Outline

Conclusion

- Adaptor Grammars (AG) “adapt” to the strings they generate
- The “rules” AGs generalise over are subtrees of base grammar
  - cached subtrees depend on data ⇒ *non-parametric*
- AGs inherit “rich get richer” property from Chinese Restaurant Processes
  - ⇒ learning *generalises over types* rather than tokens
  - ⇒ generate *Zipfian distributions*
- AGs can describe a variety of linguistic inference problems
  - Accuracy often *improves as models become more realistic*
- Sampling methods are a natural approach to AG inference

MACQUARIE UNIVERSITY

Interested in applying Bayesian inference to language?

- We're looking for:
  - *graduate students* and
  - *post-doctoral reseachers*

  with good mathematical/computational skills
- Cotutelle (joint PhD) support also available
- Please contact me if you're interested:

Prof. Mark Johnson
Department of Computing
Macquarie University
Sydney
Australia

mjohnson@science.mq.edu.au

MACQUARIE UNIVERSITY

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

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

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.

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

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

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:

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

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

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

Interested in statistical models for computational linguistics?

We're recruiting *PhD students* and *post-docs*.

Contact *Mark.Johnson@mq.edu.au* for more information.

MACQUARIE UNIVERSITY