Grammar induction in computers and people

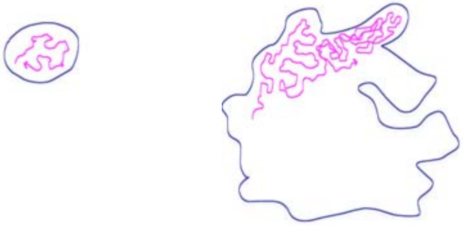
Amy Perfors
IPAM 2007

Expressivity vs. learnability

A fundamental tradeoff

Low expressivity:
easy to learn

High expressivity:
hard to learn

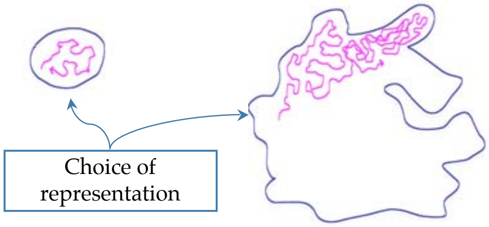


Expressivity vs. learnability

Maps onto modeling choices

Low expressivity:
easy to learn

High expressivity:
hard to learn



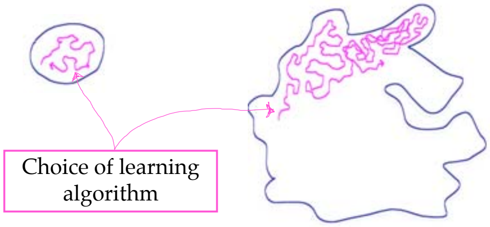
Choice of representation

Expressivity vs. learnability

Maps onto modeling choices

Low expressivity:
easy to learn

High expressivity:
hard to learn



Choice of learning algorithm

Learnability

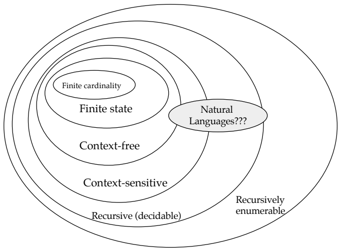
Not just about the search algorithm

- Complexity of the space
- Nature of the input
- What constitutes "having learned"

… as well as the capabilities of the learner

Gold's Theorem

Based on positive evidence only, it is impossible to learn a class of languages other than those of finite cardinality



Gold (1967)

Gold's Theorem

Based on positive evidence only, it is impossible to learn a class of languages other than those of finite cardinality

1. Listener "hears" a string
2. Listener decides whether that string is consistent with the grammar they currently think is being used (if it's the first trial, they just pick one randomly).
3. If not, generate a new grammar and go to #2. If so, stay with current grammar and go to #2.

You have learned the language once you never change grammars

Gold (1967)

Gold's Theorem

Based on positive evidence only, it is impossible to learn a class of languages other than those of finite cardinality

−

You will eventually converge: the negative evidence will rule out incorrect grammars

Gold (1967)

Gold's Theorem

Based on positive evidence only, it is impossible to learn a class of languages other than those of finite cardinality

−

You will eventually converge: the negative evidence will rule out incorrect grammars

+

If you guessed a grammar that is too large, you'll never realize it and never change

Gold (1967)

Does this lead to problems?

Kids don't seem to receive (or notice) negative evidence!

Adults tend to only correct the truth of a child's utterance, not the syntax:

Child: Mama isn't boy, he a girl.
Adult: That's right.

When adults (rarely) try to correct a child's syntax, the kid doesn't get it

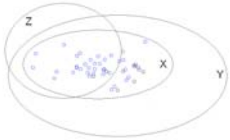
Child: Nobody don't like me.
Mother: No, say "Nobody likes me."
Child: Nobody don't like me.
[dialogue repeated 8 times]
Mother: Now listen carefully, say "NOBODY LIKES ME."
Child: Oh! Nobody don't likeS me.

Source: McNeil

Many ways to address this

(for both kids and computers)

- Change learning criterion
- Add some sort of inductive bias
  - Universal Grammar (what does this mean?)
- Change learning algorithm
  - Probabilistic learner: guaranteed in the limit (Horning, 1969)



Horning (1969), Solomonoff (1978)

But how about *not* in the limit?

Grammar induction in the real world

Grammar induction

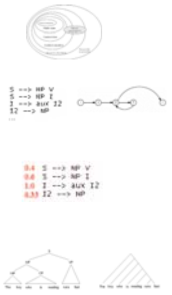
Computational question at multiple levels

* Grammar type

me ☺ ← Specific grammar structure

Mark Johnson { Parameters of specific grammar

Parsing, prediction of the dataset



Grammar induction

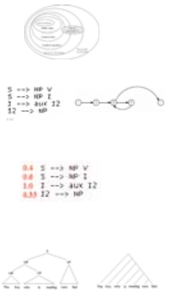
Computational question at multiple levels

Grammar type

This talk ← Specific grammar structure

Johnson, Yuille { Parameters of specific grammar

Parsing, prediction of the dataset



Plan

Unsupervised learning of grammars, by complexity

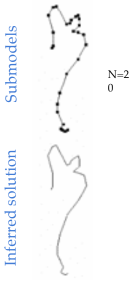
Finite-state grammars
- N-gram models (Goldwater, Griffiths, & Johnson, 2006)
  - Word segmentation and morphology
- HMMs: Bayesian model merging (Stolcke & Omohundro, 1994)
  - Phonetic learning
- HMMs: Dirichlet prior (Goldwater & Griffiths, 2007)
  - Syntactic categories

Structured grammars
- Bayesian model merging: PCFGs (Stolcke & Omohundro, 1994)
- Constituent-context model (Klein & Manning, 2002)
- Dependency grammars (Carroll & Charniak, 1992)
- Combined CCM & Dependency grammars (Klein & Manning, 2004)

Bayesian model merging: HMMs

Basic idea

- Constructed from subcomponents
  - With only a little data, the subcomponents are the datapoints themselves (plus slight generalization)
  - Construct more complex models by merging pairs of simple components
- Merging…
  - Try to do efficiently by sacrificing as little likelihood as possible each time
  - Put prior so know when to stop

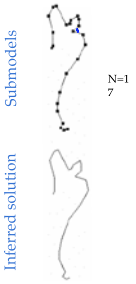


Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Basic idea

- Constructed from subcomponents
  - With only a little data, the subcomponents are the datapoints themselves (plus slight generalization)
  - Construct more complex models by merging pairs of simple components
- Merging…
  - Try to do efficiently by sacrificing as little likelihood as possible each time
  - Put prior so know when to stop

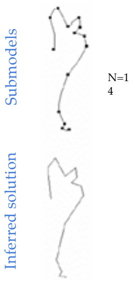


Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Basic idea

- Constructed from subcomponents
  - With only a little data, the subcomponents are the datapoints themselves (plus slight generalization)
  - Construct more complex models by merging pairs of simple components
- Merging…
  - Try to do efficiently by sacrificing as little likelihood as possible each time
  - Put prior so know when to stop

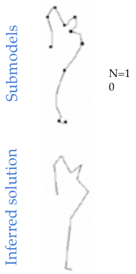


Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Basic idea

- Constructed from subcomponents
  - With only a little data, the submodels are the datapoints themselves (plus slight generalization)
  - Construct more complex models by merging pairs of simple components
- Merging…
  - Try to do efficiently by sacrificing as little likelihood as possible each time
  - Put prior so know when to stop



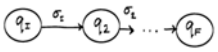
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Notation

Set of states Q

Initial state q_I, Final state q_F



Set of probability parameters:

Transition probabilities *p(q→q'):* the probability that state *q'* follows *q*

Emission probabilities *p(q | σ):* the probability that symbol *σ* is emitted when in state *q*

Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Model details

Likelihood: probability of a string *x* is the sum of the probabilities of all paths that generate *x*

$$P(x|M) = \sum_{q_1 \ldots q_l \in Q^l} p(q_I \to q_1)p(q_1 \mid x_1)p(q_1 \to q_2) \ldots p(q_l \mid x_l)p(q_l \to q_F)$$

Prior: structural vs. parameter

Parameters: Dirichlet distribution over each set of multinomial parameters (transition and emission probabilities)

$$P(\theta_M^{(q)}|M_G, M_S^{(q)}) = \frac{1}{B(\alpha_t, \ldots, \alpha_t)} \prod_{i=1}^{n_t^{(q)}} \theta_{qi}^{\alpha_t - 1} \frac{1}{B(\alpha_e, \ldots, \alpha_e)} \prod_{j=1}^{n_e^{(q)}} \theta_{qj}^{\alpha_e - 1}$$

Structural: Favor smaller number of states Q

$$P(M_S) \propto C^{-|Q|}$$

Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Algorithm

1) Construct an HMM M_0 that produces exactly the input strings
2) Loop:
   - Compute set of candidate merges K among the states of current HMM, M_i
   - For each candidate k, compute the merged model $k(M_i)$, and its posterior probability $p(k(M_i) | X)$
   - Let k^* be the merge that maximizes $p(k | M_i) | X)$. Then let $M_{i+1} = k^*(M_i)$
   - If $P(M_{i+1} | X) < P(M_i | X)$, return M_i as the induced model

Merges combine pairs of states: combined emission and transition probabilities are weighted averages of the corresponding distributions for the states that have been merged

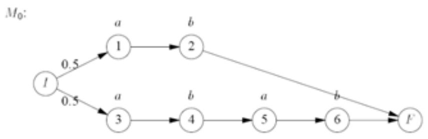
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Example

Language: *{ab}⁺*

Sentences: *ab, abab*



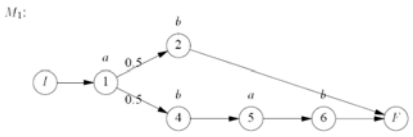
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Example

Language: *{ab}+*

Sentences: *ab, abab*



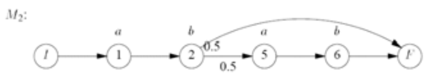
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Example

Language: *{ab}+*

Sentences: *ab, abab*



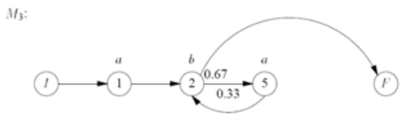
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Example

Language: *{ab}+*

Sentences: *ab, abab*



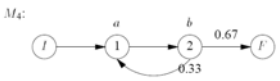
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Example

Language: *{ab}+*

Sentences: *ab, abab*



Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Why are simpler models favored?

Occam factor: Models with fewer states have fewer parameters, and therefore they make tighter predictions.

More effective data: When two states are merged, their probabilities are combined – so each state gets to effectively “see” more data

Stolcke & Omohundro, 1994

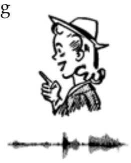
Bayesian model merging: HMMs

Application: Phonetic data

TIMIT: Collection of hand-labeled speech samples compiled for the purpose of training speaker-independent phonetic recognition systems

Contains acoustic data segmented by words and aligned with discrete labels from an alphabet of 62 phones

Goal is to construct a probabilistic model for each word in the database, representing its phonetic structure as accurately as possible



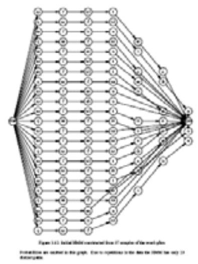
Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Application: Phonetic data

Initial HMM

Realizations of the word “often”



Stolcke & Omohundro, 1994

Bayesian model merging: HMMs

Application: Phonetic data

Learned
HMM

Realizations of
the word
“often”

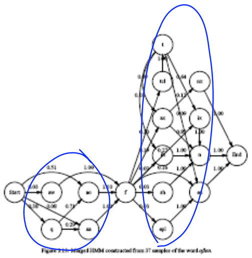


Figure 3.11: Merged HMM constructed from 37 samples of the word *often*.
Merging was constrained to keep the emission on each state unique.

Stolcke & Omohundro, 1994

More recent models of HMM learning

Application: Part-of-speech (POS) tagging

More recent models of HMM learning

Application: Part-of-speech (POS) tagging

Identifies a distribution over latent variables directly, without ever fixing particular values for the model parameters:

$$P(\mathbf{t}|\mathbf{w}) = \int P(\mathbf{t}|\mathbf{w},\theta)P(\theta|\mathbf{w})d\theta$$

w = the linguistic input; θ = parameters; t = the hidden structure

Symmetric Dirichlet prior over the transition and output distributions

$$\begin{aligned} t_i|t_{i-1}=t, t_{i-2}=t', \tau^{(t,t')} &\sim \text{Mult}(\tau^{(t,t')}) \\ w_i|t_i=t, \omega^{(t)} &\sim \text{Mult}(\omega^{(t)}) \\ \tau^{(t,t')}|\alpha &\sim \text{Dirichlet}(\alpha) \\ \omega^{(t)}|\beta &\sim \text{Dirichlet}(\beta) \end{aligned}$$

Goldwater & Griffiths, 2007

More recent models of HMM learning

POS tagging: Results

Variation of information (VI): The VI between two clusterings C (the gold standard) and C′ (the found clustering) of a set of data points is a sum of the amount of information lost in moving from C to C′

VI(C,C′) = H(C) + H(C′) – 2I(C,C′)

(High VI indicates different clustering relative to C)

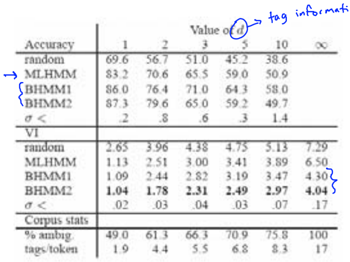
Vary the tag information given to the model: Contains tag information only for words that appear at least *d* times in the training corpus (the first 24K words of the WSJ corpus)

Goldwater & Griffiths, 2007

More recent models of HMM learning

POS tagging: Results

→ tag information



	Value of *d*					
Accuracy	1	2	3	5	10	∞
random	69.6	56.7	51.0	45.2	38.6	
→ MLHMM	83.2	70.6	65.5	59.0	50.9	
BHMM1	86.0	76.4	71.0	64.3	58.0	
BHMM2	87.3	79.6	65.0	59.2	49.7	
$\sigma <$	.2	.8	.6	.3	1.4	
VI						
random	2.65	3.96	4.38	4.75	5.13	7.29
MLHMM	1.13	2.51	3.00	3.41	3.89	6.50
BHMM1	1.09	2.44	2.82	3.19	3.47	4.30
BHMM2	**1.04**	**1.78**	**2.31**	**2.49**	**2.97**	**4.04**
$\sigma <$	.02	.03	.04	.03	.07	.17
Corpus stats						
% ambig.	49.0	61.3	66.3	70.9	75.8	100
tags/token	1.9	4.4	5.5	6.8	8.3	17

Goldwater & Griffiths, 2007

Plan

Unsupervised learning of grammars, by complexity

Finite-state grammars
- N-gram models (Goldwater, Griffiths, & Johnson, 2006)
  - Word segmentation and morphology
- HMMs: Bayesian model merging (Stolcke & Omohundro, 1994)
  - Phonetic learning
- HMMs: Dirichlet prior (Goldwater & Griffiths, 2007)
  - Syntactic categories

Structured grammars
- Bayesian model merging: PCFGs (Stolcke & Omohundro, 1994)
- Constituent-context model (Klein & Manning, 2002)
- Dependency grammars (Carroll & Charniak, 1992)
- Combined CCM & Dependency grammars (Klein & Manning, 2004)

Bayesian model merging: PCFGs

Notation

Set of nonterminal symbols *N*
Set of terminal symbols
Start nonterminal *S*
Set of productions or rules *R*
Production probabilities *p(r)* for all rules *r*

0.6	S --> NP VP
0.4	S --> NP I
0.3	NP --> N
0.4	NP --> D N
0.3	NP --> Pro

Stolcke & Omohundro, 1994

Bayesian model merging: PCFGs

Merging operator

Replaces two existing nonterminals X_1 and X_2 with a single new nonterminal Y.

Has a twofold effect on grammars:

1) RHS occurrences of X_1 and X_2 replaced by Y

$$\begin{aligned} Z_1 &\to \lambda_1 X_1 \mu_1 \quad (c_1) \\ Z_1 &\to \lambda_2 X_2 \mu_2 \quad (c_2) \\ &\Downarrow \quad \text{merge}(X_1, X_2) = Y \\ Z_1 &\to \lambda_1 Y \mu_1 \quad (c_1) \\ Z_2 &\to \lambda_2 Y \mu_2 \quad (c_2) \end{aligned}$$

2) The union of LHSs X_1 and X_2 replaced by Y

$$\begin{aligned} X_1 &\to \lambda_1 \quad (c_1) \\ X_2 &\to \lambda_2 \quad (c_2) \\ &\Downarrow \quad \text{merge}(X_1, X_2) = Y \\ Y &\to \lambda_1 \quad (c_1) \\ &\to \lambda_2 \quad (c_2) \end{aligned}$$

Stolcke & Omohundro, 1994

Bayesian model merging: PCFGs

Nonterminal chunking

Merging alone cannot create CFG productions with the usual embedding structure, so we add a chunking operator as well

Given an ordered sequence of nonterminals $X_1X_2 \ldots X_k$, create a new nonterminal *Y* that expands to $X_1X_2 \ldots X_k$, and replace occurrences of $X_1X_2 \ldots X_k$ on the RHS with *Y*

$$\begin{aligned} Z &\to \lambda X_1 X_2 \ldots X_k \mu \quad (c) \\ &\Downarrow \quad \text{chunk}(X_1 X_2 \ldots X_k) = Y \\ Z &\to \lambda Y \mu \quad (c) \\ Y &\to X_1 X_2 \ldots X_k \quad (c') \end{aligned}$$

Stolcke & Omohundro, 1994

Bayesian model merging: PCFGs

More model details

Likelihood: probability of a string *x* is the sum of the probabilities of all paths that generate *x*

$$P(x|M) = \sum_{q_1 \ldots q_l \in Q^l} p(q_I \to q_1) p(q_1 \uparrow x_1) p(q_1 \to q_2) \ldots p(q_l \uparrow x_l) p(q_l \to q_F)$$

Prior:

Parameters on productions: Dirichlet distribution over the multinomial describing each nonterminal

Structural: Description length (e.g., length of each production assumed to have been drawn from a Poisson distribution)

Stolcke & Omohundro, 1994

Bayesian model merging: PCFGs

Performance: Induction is *very* difficult

Can try to improve the greedy search in a number of ways:

It still isn't very good – the size of the space is enormous, and has many local minima

Horning (1969) – suggested an algorithm, but it was an enumeration algorithm

⇩

Structure induction for PCFGs is a very difficult problem

Stolcke & Omohundro, 1994

One possible solution: induce constituency more directly

Constituent-Context Model (CCM)

Explicitly models constituent yields and contexts

System exploits the fact that long constituents often have short, common equivalents (proforms) that appear in similar context and whose constituency is similar (or guaranteed)

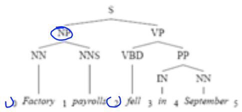
Pronoun aux adj
Det n aux adj
Det n prep det n aux adj

Model transfers the constituency of a sequence directly to its containing context, which is intended to then pressure new sequences that occur in that context into being parsed as constituents in the next round

Klein & Manning (2002)

CCM: Representation

Model captures all *spans* (contiguous subsequences) of a sentence, and each span's *context* (the terminals preceding and following it

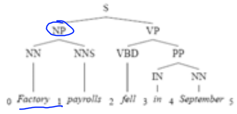


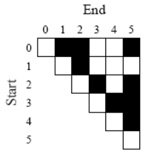
Span	Label	Constituent	Context
⟨0,5⟩	S	NN NNS VBD IN NN	⋄ – ⋄
⟨0,2⟩	NP	NN NNS	⋄ – VBD
⟨2,5⟩	VP	VBD IN NN	NNS – ⋄
⟨3,5⟩	PP	IN NN	VBD – ⋄
⟨0,1⟩	NN	NN	⋄ – NNS
⟨1,2⟩	NNS	NNS	NN – VBD
⟨2,3⟩	VBD	VBD	NNS – IN
⟨3,4⟩	IN	IN	VBD – NN
⟨4,5⟩	NN	NNS	IN – ⋄

Klein & Manning (2002)

CCM: Representation

The *bracketing* B of a sentence indicates which spans are constituents





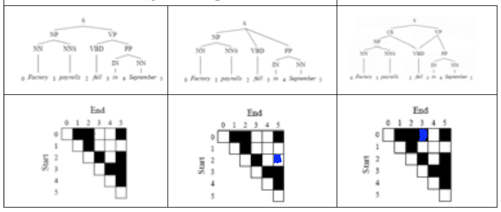
Klein & Manning (2002)

CCM: Representation

Different kinds of bracketings:

Binary

Non-crossing, Tree-equivalent

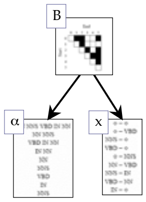


Klein & Manning (2002)

CCM: Generative Model

Choose a bracketing B according to some distribution P(B) and then generate the sentence S given B

The context and yield (content) of each span are independent of each other, and generated conditionally on the constituency of that span



$$
\begin{aligned}
P(S|B) &= \prod_{\langle i,j\rangle \in spans(S)} P(\alpha_{ij}, x_{ij} | B_{ij}) \\
&= \prod_{\langle i,j\rangle} P(\alpha_{ij}|B_{ij}) P(x_{ij}|B_{ij})
\end{aligned}
$$

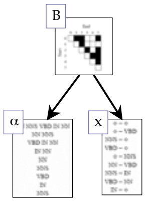
Klein & Manning (2002)

CCM: Generative Model

Choose a bracketing B according to some distribution P(B) and then generate the sentence S given B

P(B) = uniform

Soft clustering with two equal-prior classes (constituents & distituents)



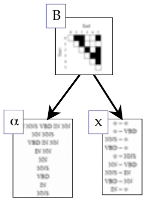
Klein & Manning (2002)

CCM: Generative Model

Choose a bracketing B according to some distribution P(B) and then generate the sentence S given B

P(B) = uniform over binary tree-equivalent bracketings

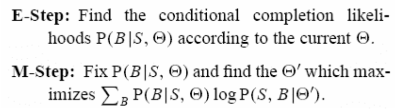
This turns distributional clustering into tree induction



Klein & Manning (2002)

CCM: Inducing structure

EM: Sentences S are observed, bracketings B are unobserved.
Parameters of the model are the constituency-conditional yield and context distributions ($P(\alpha \mid b)$ and $P(x \mid b)$, respectively)



E-Step: Find the conditional completion likelihoods $\mathrm{P}(B|S,\Theta)$ according to the current Θ.

M-Step: Fix $\mathrm{P}(B|S,\Theta)$ and find the Θ' which maximizes $\sum_B \mathrm{P}(B|S,\Theta)\log \mathrm{P}(S,B|\Theta')$.

Klein & Manning (2002)

CCM: Results

Corpus: 7422 sentences, Penn treebank Wall Street Journal that contains no more than 10 words

Evaluation: F1 (harmonic mean of precision and recall)

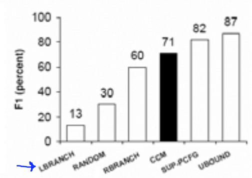


Figure 4: F_1 for various models on WSJ-10.

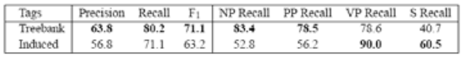
Klein & Manning (2002)

CCM: Extension

What if the model isn't given the POS tags?

Used the baseline method of word-type clustering (similar to Finch et. Al. (1993))

Performance is worse, but still better than next best (right-branching)



Tags	Precision	Recall	F_1	NP Recall	PP Recall	VP Recall	S Recall
Treebank	**63.8**	**80.2**	**71.1**	**83.4**	**78.5**	78.6	40.7
Induced	56.8	71.1	63.2	52.8	56.2	**90.0**	**60.5**

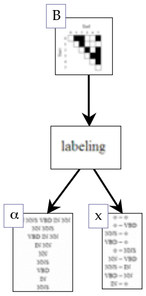
Klein & Manning (2002)

CCM: Generative Model with multiple classes

Choose a bracketing B according to some distribution P(B) and then generate the sentence S given B

Labeling is generated from that bracketing

The context and yield (content) of each span are independent of each other, and generated conditionally on the constituency of that span



Klein & Manning (2002)

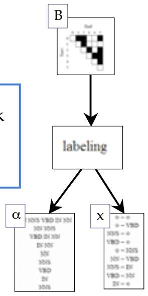
CCM: Generative Model with multiple classes

Choose a bracketing B according to some distribution P(B) and then generate the sentence S given B

Labeling is generate bracketing

This doesn't actually work that well: F1 = 70.9 (compared to 71.1)

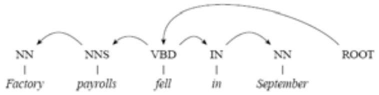
The context and yield (content) of each span are independent of each other, and generated conditionally on the constituency of that span



Klein & Manning (2002)

Another solution: use a different representation

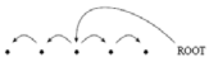
Dependency grammars



A *dependency* *d* is an arc: pair <*h*,*a*> of a head and argument, each of which is a word in a sentence *s*

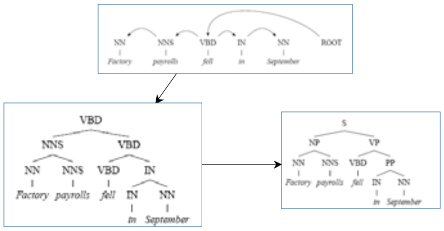
A *dependency structure* D over a sentence is a set of dependencies that form a planar, acyclic graph rooted at ROOT

Every D has an associated graph G:



Dependency grammars

Isomorphic (in terms of strong generative capacity) to a restricted form of phrase structure grammar (Miller, 1999)



Unsupervised Dependency Parsing

Why use dependency grammars?

1) Most state-of-the-art supervised parsers make use of specific lexical information in addition to word-class level information: perhaps lexical information could be a useful source of information for unsupervised models
2) A central motivation for using tree structures is to enable the extraction of dependencies, and it might be more advantageous to do so directly
3) For languages like Chinese, which have few function words, and for which the definition of lexical categories is much less clear, dependency structures may be easier to detect

Klein & Manning (2004)

Inducing dependency grammars (DEP-PCFG)

Algorithm

0) Divide the corpus into two parts, the rule corpus and the training corpus
1) For all sentences in the rule corpus, generate all rules which might be used to generate (and/or parse) the sentence
2) Estimate the probabilities for the rules
3) Using the training corpus, improve our estimate of the probabilities
4) Delete all rules with small enough probability. What remains is the grammar

Carroll & Charniak (1992)

Inducing dependency grammars (DEP-PCFG)

Difficulties

1) There are way too many possible (CFG) rules that could lead to a sentence: a potentially infinite set of nonterminals
    - That's why we use a dependency grammar: it limits it to $n(2^{n\text{-}1}+1)$, where n is length of sentence, if all terminals are distinct
2) Even with a dependency grammar, this is a LOT of sentences. For instance, a sentence with 41 terminal symbols would have

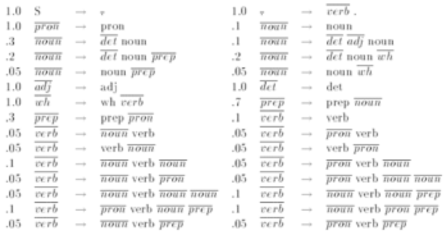
$$(41(2^{40}+1) \approx 41((2^{10})^4) \approx 40((10^3)^4) \approx 4 \cdot 10^{13}$$

3) Deal with this by ordering sentences by length (since children see simpler language before more complex language). Once a rule has been eliminated, don't consider it again

Carroll & Charniak (1992)

DEP-PCFG: Experiment

Corpus generated by artificial grammar



1.0	S	→	▾	1.0	▾	→	$\overline{verb}$.
1.0	$\overline{pron}$	→	pron	.1	$\overline{noun}$	→	noun
.3	$\overline{noun}$	→	$\overline{det}$ noun	.1	$\overline{noun}$	→	$\overline{det}$ $\overline{adj}$ noun
.2	$\overline{noun}$	→	$\overline{det}$ noun $\overline{prep}$	.2	$\overline{noun}$	→	$\overline{det}$ noun $\overline{wh}$
.05	$\overline{noun}$	→	noun $\overline{prep}$	.05	$\overline{noun}$	→	noun $\overline{wh}$
1.0	$\overline{adj}$	→	adj	1.0	$\overline{det}$	→	det
1.0	$\overline{wh}$	→	wh $\overline{verb}$	.7	$\overline{prep}$	→	prep $\overline{noun}$
.3	$\overline{prep}$	→	prep $\overline{pron}$	.1	$\overline{verb}$	→	verb
.05	$\overline{verb}$	→	$\overline{noun}$ verb	.05	$\overline{verb}$	→	$\overline{pron}$ verb
.05	$\overline{verb}$	→	verb $\overline{noun}$	.05	$\overline{verb}$	→	verb $\overline{pron}$
.1	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{noun}$	.05	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{noun}$
.05	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{pron}$	.05	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{noun}$ $\overline{noun}$
.05	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{noun}$ $\overline{noun}$	.1	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{noun}$ $\overline{prep}$
.1	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{noun}$ $\overline{prep}$	.1	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{pron}$ $\overline{prep}$
.05	$\overline{verb}$	→	$\overline{noun}$ verb $\overline{prep}$	.05	$\overline{verb}$	→	$\overline{pron}$ verb $\overline{prep}$

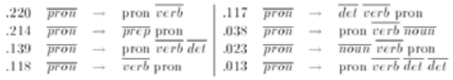
Carroll & Charniak (1992)

DEP-PCFG: Results

"Uniformly awful"

With 300 different random starting points, ended up with 300 different local minima

One segment of one grammar:



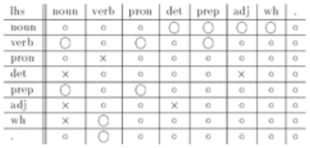
.220	$\overline{pron}$	→	pron $\overline{verb}$	.117	$\overline{pron}$	→	$\overline{det}$ $\overline{verb}$ pron
.214	$\overline{pron}$	→	$\overline{prep}$ pron	.038	$\overline{pron}$	→	pron $\overline{verb}$ $\overline{noun}$
.139	$\overline{pron}$	→	pron $\overline{verb}$ $\overline{det}$	.023	$\overline{pron}$	→	$\overline{noun}$ $\overline{verb}$ pron
.118	$\overline{pron}$	→	$\overline{verb}$ pron	.013	$\overline{pron}$	→	pron $\overline{verb}$ $\overline{det}$ $\overline{det}$

Carroll & Charniak (1992)

DEP-PCFG: Results

"Uniformly awful"

Tried to make it better by giving the grammar a chart limiting what possible nonterminals could appear on the right-hand side of certain rules



lhs	noun	verb	pron	det	prep	adj	wh	.
noun	∘	∘	∘	◯	◯	◯	◯	∘
verb	◯	∘	◯	∘	◯	∘	∘	∘
pron	∘	×	∘	∘	∘	∘	∘	∘
det	×	∘	∘	∘	∘	×	∘	∘
prep	◯	∘	◯	∘	∘	∘	∘	∘
adj	×	∘	∘	×	∘	∘	∘	∘
wh	×	◯	∘	∘	∘	∘	∘	∘
.	∘	◯	∘	∘	∘	∘	∘	∘

Figure 9: For each left-hand side, non-terminals allowed on right

It helped, but they didn't really quantify how much

Carroll & Charniak (1992)

DEP-PCFG: Results

Why didn't this work well?

1) Random initialization is not good because the parameter space is riddled with local minima; try it with a uniform initialization
2) Dependency grammars are structurally unable to distinguish order information (e.g., whether subject or object should be attached to the verb first) – and therefore they are not that great at constituency
3) It did not encode valence: for instance, can learn that nouns to the left of a verb attach to it… but if given a NOUN NOUN VERB sequence, will think that both nouns attach to the verb. Cannot encode that verbs have exactly one subject.

Carroll & Charniak (1992)

DMV: The generative model

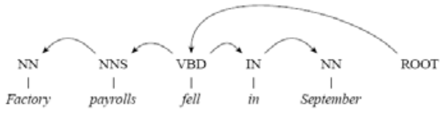
Includes a model of valence

Recursive generation process, beginning at the head h, defined according to the following rules:

For each direction L and R from h

- With P(STOP | h,dir,adj) stop generating additional arguments
- If don't stop, generate a new argument a with P(a | h,dir)P(D(a))

→ These probability factors are the model's parameters



Klein & Manning (2004)

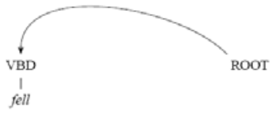
DMV: The generative model

Recursive generation process, beginning at the head h, defined according to the following rules:

For each direction L and R from h

- With P(STOP | h,dir,adj) stop generating additional arguments
- If don't stop, generate a new argument a with P(a | h,dir)P(D(a))

→ These probability factors are the model's parameters



Klein & Manning (2004)

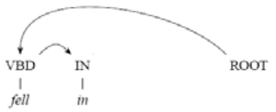
DMV: The generative model

Recursive generation process, beginning at the head *h*, defined according to the following rules:

For each direction L and R from *h*

- With P(STOP | *h,dir,adj*) stop generating additional arguments
- If don't stop, generate a new argument a with P(*a* | *h,dir*)P(D(*a*))

→ These probability factors are the model's parameters



Klein & Manning (2004)

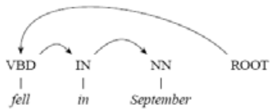
DMV: The generative model

Recursive generation process, beginning at the head *h*, defined according to the following rules:

For each direction L and R from *h*

- With P(STOP | *h,dir,adj*) stop generating additional arguments
- If don't stop, generate a new argument a with P(*a* | *h,dir*)P(D(*a*))

→ These probability factors are the model's parameters



Klein & Manning (2004)

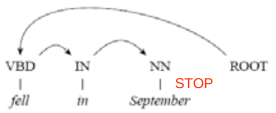
DMV: The generative model

Recursive generation process, beginning at the head *h*, defined according to the following rules:

For each direction L and R from *h*

- With P(STOP | *h,dir,adj*) stop generating additional arguments
- If don't stop, generate a new argument a with P(*a* | *h,dir*)P(D(*a*))

→ These probability factors are the model's parameters



Klein & Manning (2004)

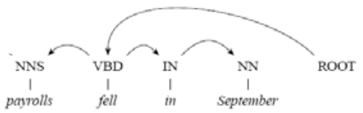
DMV: The generative model

Recursive generation process, beginning at the head *h*, defined according to the following rules:

For each direction L and R from *h*

- With P(STOP | *h,dir,adj*) stop generating additional arguments
- If don't stop, generate a new argument a with P(*a* | *h,dir*)P(D(*a*))

→ These probability factors are the model's parameters



Klein & Manning (2004)

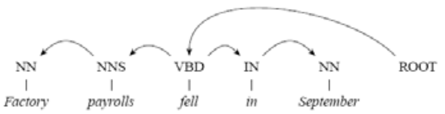
DMV: The generative model

Recursive generation process, beginning at the head *h*, defined according to the following rules:

For each direction L and R from *h*

- With P(STOP | *h,dir,adj*) stop generating additional arguments
- If don't stop, generate a new argument a with P(*a* | *h,dir*)P(D(*a*))

→ These probability factors are the model's parameters



Klein & Manning (2004)

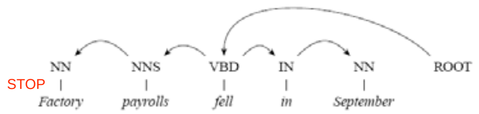
DMV: The generative model

Recursive generation process, beginning at the head *h*, defined according to the following rules:

For each direction L and R from *h*

- With P(STOP | *h,dir,adj*) stop generating additional arguments
- If don't stop, generate a new argument *a* with P(*a* | *h,dir*)P(D(*a*))

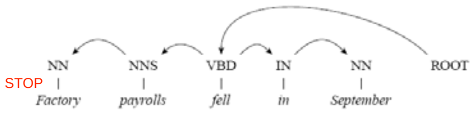
→ These probability factors are the model's parameters



Klein & Manning (2004)

DMV: The generative model

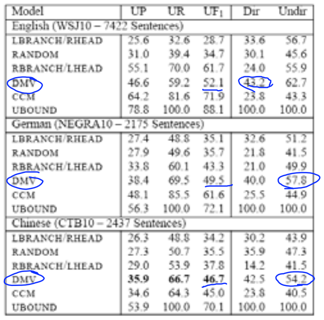
Use inside-outside algorithm for reestimation of the parameters (the probability factors for STOP and CHOOSE)



Klein & Manning (2004)

DMV: Experiment

F_1



Model	UP	UR	UF$_1$	Dir	Undir
English (WSJ10 – 7422 Sentences)					
LBRANCH/RHEAD	25.6	32.6	28.7	33.6	56.7
RANDOM	31.0	39.4	34.7	30.1	45.6
RBRANCH/LHEAD	55.1	70.0	61.7	24.0	55.9
DMV	46.6	59.2	52.1	43.2	62.7
CCM	64.2	81.6	71.9	23.8	43.3
UBOUND	78.8	100.0	88.1	100.0	100.0
German (NEGRA10 – 2175 Sentences)					
LBRANCH/RHEAD	27.4	48.8	35.1	32.6	51.2
RANDOM	27.9	49.6	35.7	21.8	41.5
RBRANCH/LHEAD	33.8	60.1	43.3	21.0	49.9
DMV	38.4	69.5	49.5	40.0	57.8
CCM	48.1	85.5	61.6	25.5	44.9
UBOUND	56.3	100.0	72.1	100.0	100.0
Chinese (CTB10 – 2437 Sentences)					
LBRANCH/RHEAD	26.3	48.8	34.2	30.2	43.9
RANDOM	27.3	50.7	35.5	35.9	47.3
RBRANCH/LHEAD	29.0	53.9	37.8	14.2	41.5
DMV	**35.9**	**66.7**	**46.7**	42.5	54.2
CCM	34.6	64.3	45.0	23.8	40.5
UBOUND	53.9	100.0	70.1	100.0	100.0

Klein & Manning (2004)

Why not combine models?

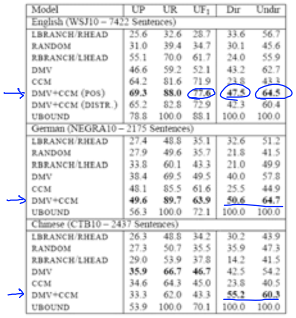
Their strengths are complementary

Both CCM and DMV can be seen as models over lexicalized trees

Combine them by scoring each tree with the product of all the probabilities from the individual models

Klein & Manning (2004)

DMV-CCM: Results



Model	UP	UR	UF$_1$	Dir	Undir
English (WSJ10 – 7422 Sentences)					
LBRANCH/RHEAD	25.6	32.6	28.7	33.6	56.7
RANDOM	31.0	39.4	34.7	30.1	45.6
RBRANCH/LHEAD	55.1	70.0	61.7	24.0	55.9
DMV	46.6	59.2	52.1	43.2	62.7
CCM	64.2	81.6	71.9	23.8	43.3
→ DMV+CCM (POS)	**69.3**	**88.0**	**77.6**	**47.5**	**64.5**
DMV+CCM (DISTR.)	65.2	82.8	72.9	42.3	60.4
UBOUND	78.8	100.0	88.1	100.0	100.0
German (NEGRA10 – 2175 Sentences)					
LBRANCH/RHEAD	27.4	48.8	35.1	32.6	51.2
RANDOM	27.9	49.6	35.7	21.8	41.5
RBRANCH/LHEAD	33.8	60.1	43.3	21.0	49.9
DMV	38.4	69.5	49.5	40.0	57.8
CCM	48.1	85.5	61.6	25.5	44.9
→ DMV+CCM	**49.6**	**89.7**	**63.9**	**50.6**	**64.7**
UBOUND	56.3	100.0	72.1	100.0	100.0
Chinese (CTB10 – 2437 Sentences)					
LBRANCH/RHEAD	26.3	48.8	34.2	30.2	43.9
RANDOM	27.3	50.7	35.5	35.9	47.3
RBRANCH/LHEAD	29.0	53.9	37.8	14.2	41.5
DMV	**35.9**	**66.7**	**46.7**	42.5	54.2
CCM	34.6	64.3	45.0	23.8	40.5
→ DMV+CCM	33.3	62.0	43.3	**55.2**	**60.3**
UBOUND	53.9	100.0	70.1	100.0	100.0

Klein & Manning (2004)

Conclusion

Grammar induction (of structure) is difficult

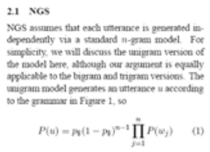
- Expressivity/learnability tradeoff
- Finite-state grammars are easier, and there are some useful linguistic domains that they are reasonable models for
  - Word segmentation
  - Phonetics
  - Syntactic categories
- More structured grammars are difficult
  - As we saw in Mark Johnson's talk, PCFGs aren't great models for language, but even they are quite hard
  - Constituent-context model
  - Dependency grammars

N-gram models
(word segmentation)

Hist / overview

Two
successful word segmentation systems based on explicit probabilistic models are those of Brent (1999) and Venkataraman (2001). Brent's Model-Based Dynamic Programming (MBDP) system assumes a unigram word distribution. Venkataraman uses standard unigram, bigram, and trigram language models in three versions of his system, which we refer to as n-gram Segmentation (NGS).

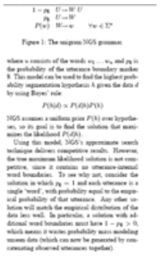
N-gram segmentation (NGS)



2.1 NGS

NGS assumes that each utterance is generated independently via a standard n-gram model. For simplicity, we will discuss the unigram version of the model here, although our argument is equally applicable to the bigram and trigram versions. The unigram model generates an utterance u according to the grammar in Figure 1, so

$$P(u) = p_\$(1-p_\$)^{n-1}\prod_{j=1}^{n} P(w_j) \qquad (1)$$



$1-p_\$$	$U \rightarrow W\ U$	
$p_\$$	$U \rightarrow W$	
$P(w)$	$W \rightarrow w$	$\forall w \in \Sigma^*$

Figure 1: The unigram NGS grammar.

where u consists of the words $w_1 \ldots w_n$ and $p_\$$ is the probability of the utterance boundary marker \$. This model can be used to find the highest probability segmentation hypothesis h given the data d by using Bayes' rule:

$$P(h|d) \propto P(d|h)P(h)$$

NGS assumes a uniform prior $P(h)$ over hypotheses, so its goal is to find the solution that maximizes the likelihood $P(d|h)$.

Using this model, NGS's approximate search technique delivers competitive results. However, the true maximum likelihood solution is not competitive, since it contains no utterance-internal word boundaries. To see why not, consider the solution in which $p_\$ = 1$ and each utterance is a single 'word', with probability equal to the empirical probability of that utterance. Any other solution will match the empirical distribution of the data less well. In particular, a solution with additional word boundaries must have $1 - p_\$ > 0$, which means it wastes probability mass modeling unseen data (which can now be generated by concatenating observed utterances together).

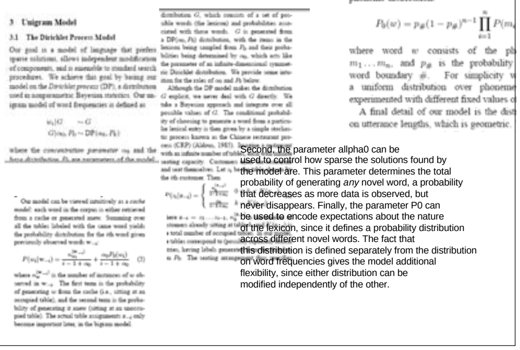
Intuitively, the NGS model considers the unsegmented solution to be optimal because it ranks all hypotheses equally probable *a priori*. We know, however, that hypotheses that memorize the input data are unlikely to generalize to unseen data, and are therefore poor solutions. To prevent memorization, we could restrict our hypothesis space to models with fewer parameters than the number of utterances in the data. A more general and mathematically satisfactory solution is to assume a nonuniform prior, assigning higher probability to hypotheses with fewer parameters. This is in fact the route taken by Brent in his MBDP model, as we shall see in the following section.

Model-based dynamic programming (Brent) MBDP

MBDP assumes a corpus of utterances is generated
as a single probabilistic event with four steps:
1. Generate L, the number of lexical types.
2. Generate a phonemic representation for each
type (except the utterance boundary type, $).
3. Generate a token frequency for each type.
4. Generate an ordering for the set of tokens.
In a final deterministic step, the ordered tokens
are concatenated to create an unsegmented corpus.
This means that certain segmented corpora
will produce the observed data with probability 1,
and all others will produce it with probability 0.
The posterior probability of a segmentation given
the data is thus proportional to its prior probability
under the generative model, and the best segmentation
is that with the highest prior probability.
There are two important points to note about
the MBDP model. First, the distribution over L
assigns higher probability to models with fewer
lexical items. We have argued that this is necessary
to avoid memorization, and indeed the unsegmented
corpus is not the optimal solution under
this model, as we will show in Section 3. Second,
the factorization into four separate steps makes
it theoretically possible to modify each step independently
in order to investigate the effects of
the various modeling assumptions. However, the
mathematical statement of the model and the approximations
necessary for the search procedure
make it unclear how to modify the model in any
interesting way. In particular, the fourth step uses
a uniform distribution, which creates a unigram
constraint that cannot easily be changed.

Prof don't want to go over this - v. confusing - suffice to know Brent assumes prior (as they do) thus solves prob from doing MLE

Dirichlet model



3 Unigram Model

3.1 The Dirichlet Process Model

Our goal is a model of language that prefers sparse solutions, allows independent modification of components, and is amenable to standard search procedures. We achieve this goal by basing our model on the *Dirichlet process* (DP), a distribution used in nonparametric Bayesian statistics. Our unigram model of word frequencies is defined as

$$w_i|G \sim G$$
$$G|\alpha_0, P_0 \sim \mathrm{DP}(\alpha_0, P_0)$$

where the *concentration parameter* α_0 and the *base distribution* P_0 are parameters of the model.

distribution G, which consists of a set of possible words (the lexicon) and probabilities associated with those words. G is generated from a DP(α_0, P_0) distribution, with the items in the lexicon being sampled from P_0 and their probabilities being determined by α_0, which acts like the parameter of an infinite-dimensional symmetric Dirichlet distribution. We provide some intuition for the roles of α_0 and P_0 below.

Although the DP model makes the distribution G explicit, we never deal with G directly. We take a Bayesian approach and integrate over all possible values of G. The conditional probability of choosing to generate a word from a particular lexical entry is then given by a simple stochastic process known as the Chinese restaurant process (CRP) (Aldous, 1985). Imagine a restaurant with an infinite number of tables, [illegible] seating capacity. Customers [illegible] and seat themselves. Let z_i be [illegible] the ith customer. Then

$$P(z_i|\mathbf{z}_{-i}) = \begin{cases} \frac{n_k^{(\mathbf{z}_{-i})}}{i-1+\alpha_0} & [illegible] \\ \frac{\alpha_0}{i-1+\alpha_0} & [illegible] \end{cases}$$

here $\mathbf{z}_{-i} = z_1 \dots z_{i-1}$, $n_k^{(\mathbf{z}_{-i})}$ [illegible] customers already sitting at table k, and $K(\mathbf{z}_{-i})$ [illegible] total number of occupied tables. In our model, [illegible] tables correspond to (possibly repeated) lexical entries, having labels generated [illegible] P_0. The seating arrangement thus specifies [illegible]

Our model can be viewed intuitively as a *cache model*: each word in the corpus is either retrieved from a cache or generated anew. Summing over all the tables labeled with the same word yields the probability distribution for the ith word given previously observed words $\mathbf{w}_{-i}$:

$$P(w_i|\mathbf{w}_{-i}) = \frac{n_{w_i}^{(\mathbf{w}_{-i})}}{i-1+\alpha_0} + \frac{\alpha_0 P_0(w_i)}{i-1+\alpha_0} \quad (3)$$

where $n_w^{(\mathbf{w}_{-i})}$ is the number of instances of w observed in $\mathbf{w}_{-i}$. The first term is the probability of generating w from the cache (i.e., sitting at an occupied table), and the second term is the probability of generating it anew (sitting at an unoccupied table). The actual table assignments $\mathbf{z}_{-i}$ only become important later, in the bigram model.

Since the goal of this paper is to investi[illegible] role of context in word segmentation, w[illegible] the simplest possible model for P_0, i.e. a [illegible] phoneme distribution:

$$P_0(w) = p_\#(1-p_\#)^{n-1}\prod_{i=1}^{n} P(m_{[illegible]}$$

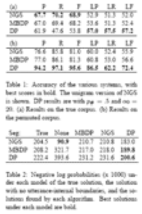
where word w consists of the ph[illegible] $m_1 \dots m_n$, and $p_\#$ is the probability [illegible] word boundary #. For simplicity w[illegible] a uniform distribution over phoneme[illegible] experimented with different fixed values o[illegible]

A final detail of our model is the dis[illegible] on utterance lengths, which is geometric.

Second, the parameter allpha0 can be
used to control how sparse the solutions found by
the model are. This parameter determines the total
probability of generating *any* novel word, a probability
that decreases as more data is observed, but
never disappears. Finally, the parameter P0 can
be used to encode expectations about the nature
of the lexicon, since it defines a probability distribution
across different novel words. The fact that
this distribution is defined separately from the distribution
on word frequencies gives the model additional
flexibility, since either distribution can be
modified independently of the other.

Accuracy of unigram model

In Table 1(a), we compare the results of our system
to those of MBDP and NGS.4 Although our
system has higher lexicon accuracy than the others,
its token accuracy is much worse. This result
occurs because our system often mis-analyzes frequently
occurring words. In particular, many of
these words occur in common collocations such
as *what's that* and *do you*, which the system interprets
as a single words. It turns out that a full 31%
of the proposed lexicon and nearly 30% of tokens
consist of these kinds of errors.
Upon reflection, it is not surprising that a unigram
language model would segment words in this
way. Collocations violate the unigram assumption
in the model, since they exhibit strong word-toword
dependencies. The only way the model can
capture these dependencies is by assuming that
these collocations are in fact words themselves.
Why don't the MBDP and NGS unigram models
exhibit these problems? We have already
shown that NGS's results are due to its search procedure
rather than its model. The same turns out
to be true for MBDP. Table 2 shows the probabilities
under each model of various segmentations of
the corpus. From these figures, we can see that
the MBDP model assigns higher probability to the
solution found by our Gibbs sampler than to the
solution found by Brent's own incremental search
algorithm. In other words, Brent's model *does* prefer
the lower-accuracy collocation solution, but his
search algorithm instead finds a higher-accuracy
but lower-probability solution.



(a)	P	R	F	LP	LR	LF
NGS	**67.7**	**70.2**	**68.9**	52.9	51.3	52.0
MBDP	67.0	69.4	68.2	53.6	51.3	52.4
DP	61.9	47.6	53.8	**57.0**	**57.5**	**57.2**

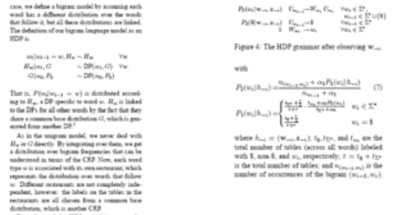
(b)	P	R	F	LP	LR	LF
NGS	76.6	85.8	81.0	60.0	52.4	55.9
MBDP	77.0	86.1	81.3	60.8	53.0	56.6
DP	**94.2**	**97.1**	**95.6**	**86.5**	**62.2**	**72.4**

Table 1: Accuracy of the various systems, with best scores in bold. The unigram version of NGS is shown. DP results are with $p_\# = .5$ and $\alpha_0 = 20$. (a) Results on the true corpus. (b) Results on the permuted corpus.

Seg:	True	None	MBDP	NGS	DP
NGS	204.5	**90.9**	210.7	210.8	183.0
MBDP	208.2	321.7	217.0	218.0	**189.8**
DP	222.4	393.6	231.2	231.6	**200.6**

Table 2: Negative log probabilities (x 1000) under each model of the true solution, the solution with no utterance-internal boundaries, and the solutions found by each algorithm. Best solutions under each model are bold.

Bigram model



case, we define a bigram model by assuming each word has a different distribution over the words that follow it, but all these distributions are linked. The definition of our bigram language model as an HDP is

$$w_i|w_{i-1} = w, H_w \sim H_w \quad \forall w$$
$$H_w|\alpha_1, G \sim \mathrm{DP}(\alpha_1, G) \quad \forall w$$
$$G|\alpha_0, P_0 \sim \mathrm{DP}(\alpha_0, P_0)$$

That is, $P(w_i|w_{i-1} = w)$ is distributed according to H_w, a DP specific to word w. H_w is linked to the DPs for all other words by the fact that they share a common base distribution G, which is generated from another DP.[7]

As in the unigram model, we never deal with H_w or G directly. By integrating over them, we get a distribution over bigram frequencies that can be understood in terms of the CRP. Now, each word type w is associated with its own restaurant, which represents the distribution over words that follow w. Different restaurants are not completely independent, however: the labels on the tables in the restaurants are all chosen from a common base distribution, which is another CRP.

$P_2(w_i|\mathbf{w}_{-i}, \mathbf{z}_{-i})$ $U_{w_{i-1}} \rightarrow W_{w_i}\, U_{w_i}$ $\forall w_i \in \Sigma^*, w_{i-1} \in \Sigma^* \cup \{\$\}$
$P_2(\$|\mathbf{w}_{-i}, \mathbf{z}_{-i})$ $U_{w_{i-1}} \rightarrow \$$ $\forall w_{i-1} \in \Sigma^*$
1 $W_{w_i} \rightarrow w_i$ $\forall w_i \in \Sigma^*$

Figure 4: The HDP grammar after observing $\mathbf{w}_{-i}$.

with

$$P_2(w_i|h_{-i}) = \frac{n_{(w_{i-1},w_i)} + \alpha_1 P_1(w_i|h_{-i})}{n_{w_{i-1}} + \alpha_1} \quad (7)$$

$$P_1(w_i|h_{-i}) = \begin{cases} \frac{t_{\Sigma^*}+\frac{\tau}{2}}{t+\tau} \cdot \frac{t_{w_i}+\alpha_0 P_0(w_i)}{t_{\Sigma^*}+\alpha_0} & w_i \in \Sigma^* \\ \frac{t_\$+\frac{\tau}{2}}{t+\tau} & w_i = \$ \end{cases}$$

where $h_{-i} = (\mathbf{w}_{-i}, \mathbf{z}_{-i})$; $t_\$$, t_{Σ^*}, and t_{w_i} are the total number of tables (across all words) labeled with \$, non-\$, and w_i, respectively; $t = t_\$ + t_{\Sigma^*}$ is the total number of tables; and $n_{(w_{i-1},w_i)}$ is the number of occurrences of the bigram (w_{i-1}, w_i).

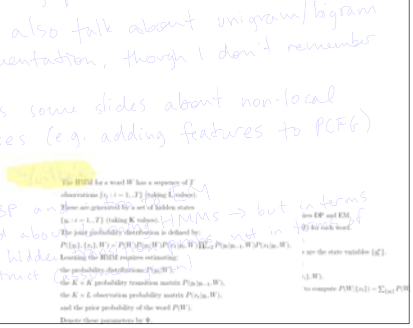
[illegible]

- Introduced HMMs, PCFGs, formalizm (a bit on Chomsky hierarchy but not much)
- Talked about how to parse sentences w/PCFGs or use to predict sentences
  - and how to estimate params from the data (EM, inside-outside)
- Unsupervised inference for θ (and thus parse trees)
  - Bayesian prior on θ
  - in a way is learning grammar (as in Sesotho example) but still very constrained

Then he looks at grammar induction
- well, sorta; prior on θ makes it sort of like also choosing prods

His slides also talk about unigram/bigram word segmentation, though I don't remember it

He also has some slides about non-local dependencies (e.g. adding features to PCFG)



The HMM for a word W has a sequence of T observations $\{x_i : i = 1,..,T\}$ (taking L values).
These are generated by a set of hidden states $\{y_i : i = 1,..,T\}$ (taking K values).
The joint probability distribution is defined by:
$P(\{y_i\},\{x_i\},W) = P(W)P(y_1|W)P(x_1|y_1,W)\prod_{i=2}^{T} P(y_i|y_{i-1},W)P(x_i|y_i,W)$.
Learning the HMM requires estimating:
the probability distributions $P(y_1|W)$,
the $K \times K$ probability transition matrix $P(y_i|y_{i-1},W)$,
the $K \times L$ observation probability matrix $P(x_i|y_i,W)$,
and the prior probability of the word $P(W)$.
Denote these parameters by Φ.

[illegible] DP and EM.
[illegible] for each word.
[illegible] are the state variables $\{y_i^t\}$.
[illegible] $\},W)$.
[illegible] to compute $P(W|\{x_i\}) = \sum_{\{y_i\}} P(W$

- Did BP and intro to EM
- Talked about learning HMMs → but in terms of transition probs, not in terms of hidden struct (assumed given)

Problems with PCFGs

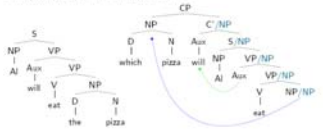
from Johnson

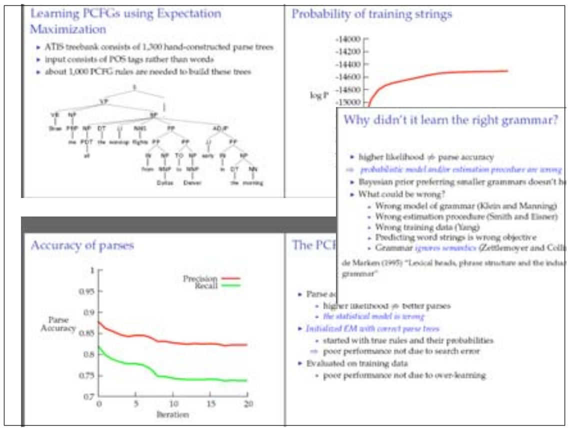
Nonlocal "movement" constructions

"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)





Learning PCFGs using Expectation Maximization

- 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

Why didn't it learn the right grammar?

- higher likelihood ⇏ parse accuracy

⇒ *probabilistic model and/or estimation procedure are wrong*

- Bayesian prior preferring smaller grammars doesn't h
- What could be wrong?
  - Wrong model of grammar (Klein and Manning)
  - Wrong estimation procedure (Smith and Eisner)
  - Wrong training data (Yang)
  - Predicting word strings is wrong objective
  - Grammar *ignores semantics* (Zettlemoyer and Colli

de Marken (1995) "Lexical heads, phrase structure and the induc grammar"

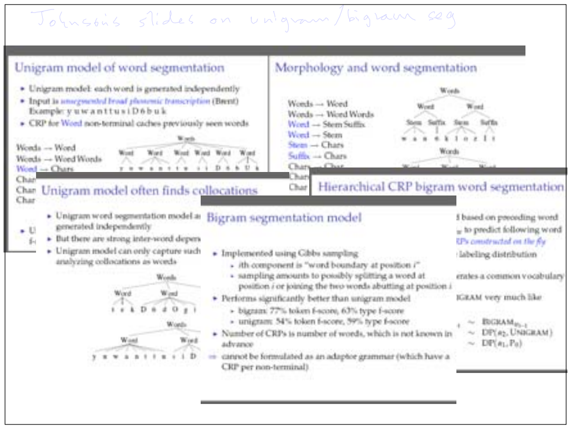
Accuracy of parses

The PCF

- Parse ac
  - higher likelihood ⇏ better parses
  - *the statistical model is wrong*
- *Initialized EM with correct parse trees*
  - started with true rules and their probabilities

  ⇒ poor performance not due to search error
- Evaluated on training data
  - poor performance not due to over-learning

Johnson's slides on unigram/bigram seg



Unigram model of word segmentation

- Unigram model: each word is generated independently
- Input is *unsegmented broad phonemic transcription* (Brent)
  Example: y u w a n t t u s i D 6 b u k
- CRP for Word non-terminal caches previously seen words

Words → Word
Words → Word Words
Word → Chars
Char
Char
Char

- U
  f-

Morphology and word segmentation

Words → Word
Words → Word Words
Word → Stem Suffix
Word → Stem
Stem → Chars
Suffix → Chars
Chars → Char
Chars
Char

Unigram model often finds collocations

- Unigram word segmentation model a generated independently
- But there are strong inter-word deper
- Unigram model can only capture such analyzing collocations as words

Hierarchical CRP bigram word segmentation

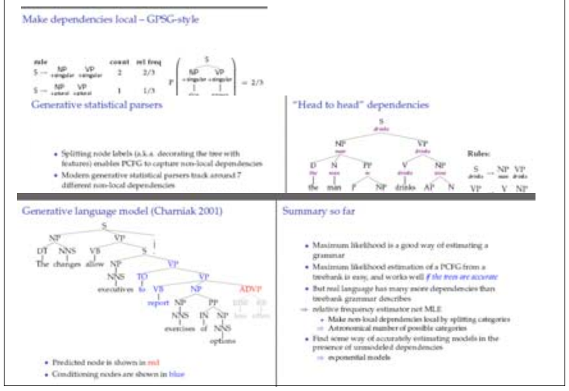
d based on preceding word
to predict following word
RPs constructed on the fly
labeling distribution
erates a common vocabulary
IGRAM very much like
~ BIGRAM
~ DP(α2, UNIGRAM)
~ DP(α1, P0)

Bigram segmentation model

- Implemented using Gibbs sampling
  - *i*th component is "word boundary at position *i*"
  - sampling amounts to possibly splitting a word at position *i* or joining the two words abutting at position *i*
- Performs significantly better than unigram model
  - bigram: 77% token f-score, 63% type f-score
  - unigram: 54% token f-score, 59% type f-score
- Number of CRPs is number of words, which is not known in advance

⇒ cannot be formulated as an adaptor grammar (which have a CRP per non-terminal)

Johnson's slides on adding to PCFGs



Make dependencies local – GPSG-style

rule	count	rel freq
S → NP singular VP singular	2	2/3
S → NP plural VP plural	1	1/3

P (S → NP +singular VP +singular) = 2/3

Generative statistical parsers

- Splitting node labels (a.k.a. decorating the tree with features) enables PCFG to capture non-local dependencies
- Modern generative statistical parsers track around 7 different non-local dependencies

"Head to head" dependencies

Rules:

S drinks → NP man VP drinks

VP → V NP

Generative language model (Charniak 2001)

- Predicted node is shown in red
- Conditioning nodes are shown in blue

Summary so far

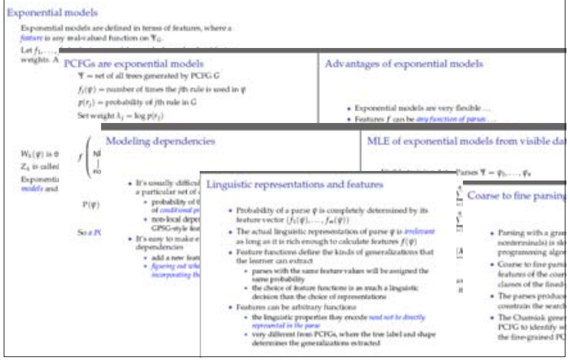
- Maximum likelihood is a good way of estimating a grammar
- Maximum likelihood estimation of a PCFG from a treebank is easy, and works well *if the trees are accurate*
- But real language has many more dependencies than treebank grammar describes

⇒ relative frequency estimator not MLE
  - Make non-local dependencies local by splitting categories

  ⇒ Astronomical number of possible categories
- Find some way of accurately estimating models in the presence of unmodeled dependencies

  ⇒ exponential models

More extensions of PCFGs



Exponential models

Exponential models are defined in terms of features, where a *feature* is any real-valued function on Ψ_G.

Let $f_1, \ldots,$
weights. A

$W_\lambda(\psi)$ is th
Z_λ is called
Exponenti
models and

PCFGs are exponential models

Ψ = set of all trees generated by PCFG G

$f_j(\psi)$ = number of times the jth rule is used in ψ

$p(r_j)$ = probability of jth rule in G

Set weight $\lambda_j = \log p(r_j)$

$P(\psi)$

So *a PC*

Advantages of exponential models

- Exponential models are very flexible ...
- Features f can be *any function of parses* ...

Modeling dependencies

- It's usually difficul a particular set of
  - probability of t of *conditional pr*
  - non-local depe GPSG-style fea
- It's easy to make e dependencies
  - add a new feat
  - *figuring out wh incorporating th*

MLE of exponential models from visible da

Parses $\Psi = \psi_1, \ldots, \psi_n$

Linguistic representations and features

- Probability of a parse ψ is completely determined by its feature vector $(f_1(\psi), \ldots, f_m(\psi))$
- The actual linguistic representation of parse ψ is *irrelevant* as long as it is rich enough to calculate features $f(\psi)$
- Feature functions define the kinds of generalizations that the learner can extract
  - parses with the same feature values will be assigned the same probability
  - the choice of feature functions is as much a linguistic decision than the choice of representations
- Features can be arbitrary functions
  - the linguistic properties they encode *need not be directly represented in the parse*
  - very different from PCFGs, where the tree label and shape determines the generalizations extracted

Coarse to fine parsing

- Parsing with a gran nonterminals) is slo programming algo
- Coarse to fine parsi features of the coar classes of the fined
- The parses produc constrain the search
- The Charniak gene PCFG to identify w the fine-grained PC