Statistical models for natural language parsing

Mark Johnson

Microsoft Research / Brown University

July 2007

Outline

Introduction

Non-local dependencies and the PCFG MLE

Generative statistical parsers

Exponential (a.k.a. Maximum Entropy) parsing models

Coarse to fine reranking

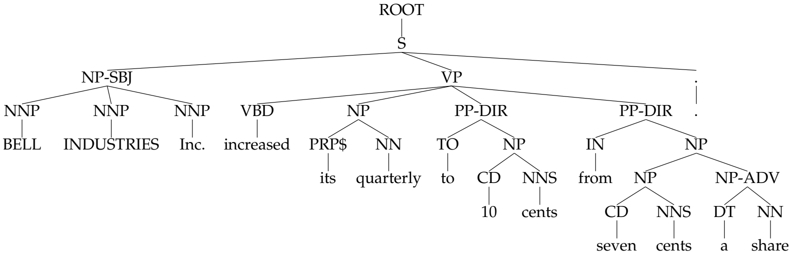
Self-training of the reranking parser

Sample parser errors

Preview of natural language parsing

- Non-local dependencies cause PCFG Maximum Likelihood Estimator (MLE) to produce sub-optimal grammars
- State-splitting or decorating with features can make non-local dependencies local
- Exponential (a.k.a. Maximum Entropy) models aren't as adversely affected by non-local dependencies as PCFGs
- But MLE seems difficult to compute $\Rightarrow$ Maximum Conditional Likelihood Estimation (MCLE)
- MCLE also seems better suited to parsing tasks if PCFG doesn't accurately describe distribution of strings
- Coarse-to-fine reranking combines PCFG and exponential models to produce the most accurate parsers we have today

Treebank corpora



- The Penn treebank contains hand-annotated parse trees for $\sim 50,000$ sentences
- Treebanks also exist for the Brown corpus, the Switchboard corpus (spontaneous telephone conversations) and Chinese and Arabic corpora

Outline

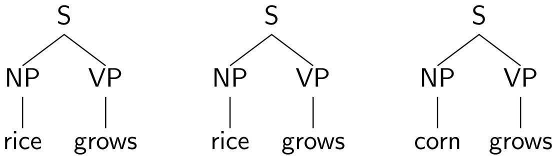
Estimating a grammar from a treebank

- *Maximum likelihood principle*: Choose the grammar and rule probabilities that make the trees in the corpus *as likely as possible*
  - read the rules off the trees
  - for PCFGs, set rule probabilities to the *relative frequency* of each rule in the treebank

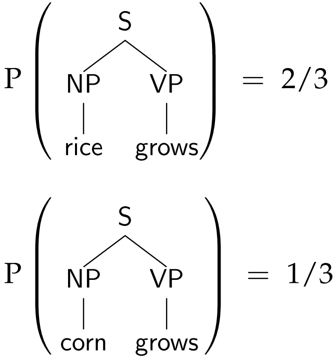
$$\mathrm{P}(\mathrm{VP} \rightarrow \mathrm{V\ NP}) = \frac{\text{Number of times VP} \rightarrow \text{V NP occurs}}{\text{Number of times VP occurs}}$$

- *If the language is generated by a PCFG and the treebank trees are its derivation trees, the estimated grammar converges to the true grammar.*

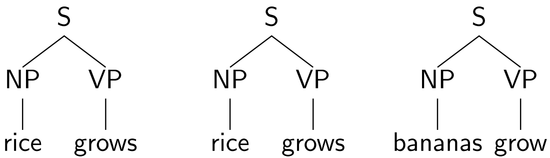
Estimating PCFGs from visible data



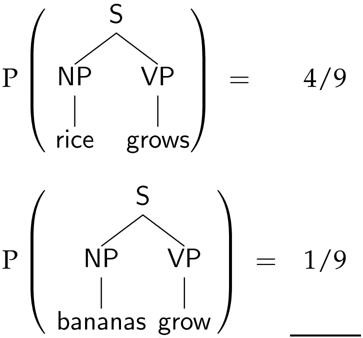
Rule	**Count**	**Rel Freq**
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	2/3
NP $\rightarrow$ corn	1	1/3
VP $\rightarrow$ grows	3	1



Non-local dependencies and PCFG MLE

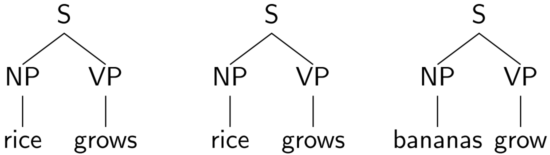


Rule	**Count**	**Rel Freq**
S → NP VP	3	1
NP → rice	2	2/3
NP → bananas	1	1/3
VP → grows	2	2/3
VP → grow	1	1/3

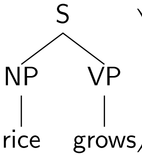


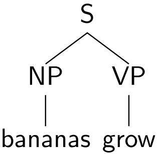
partition function $Z = 5/9$

Dividing by partition function Z



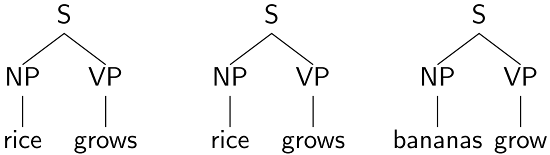
Rule	Count	Rel Freq
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	$2/3$
NP $\rightarrow$ bananas	1	$1/3$
VP $\rightarrow$ grows	2	$2/3$
VP $\rightarrow$ grow	1	$1/3$

P () $= \quad 4/9 \quad 4/5$

P () $= \quad 1/9 \quad 1/5$

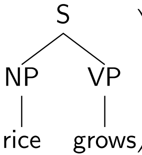
$Z = 5/9$

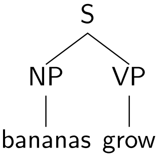
Other values do better!



Rule	**Count**	**p**
S → NP VP	3	1
NP → rice	2	2/3
NP → bananas	1	1/3
VP → grows	2	1/2
VP → grow	1	1/2

(Abney 1997)

P () = 2/6 2/3

P () = 1/6 1/3

$Z = 3/6$

Make dependencies local – GPSG-style

rule	count	rel freq
$\mathsf{S} \rightarrow \underset{+\mathsf{singular}}{\mathsf{NP}}\ \underset{+\mathsf{singular}}{\mathsf{VP}}$	2	2/3
$\mathsf{S} \rightarrow \underset{+\mathsf{plural}}{\mathsf{NP}}\ \underset{+\mathsf{plural}}{\mathsf{VP}}$	1	1/3
$\underset{+\mathsf{singular}}{\mathsf{NP}} \rightarrow \mathsf{rice}$	2	1
$\underset{+\mathsf{plural}}{\mathsf{NP}} \rightarrow \mathsf{bananas}$	1	1
$\underset{+\mathsf{singular}}{\mathsf{VP}} \rightarrow \mathsf{grows}$	2	1
$\underset{+\mathsf{plural}}{\mathsf{VP}} \rightarrow \mathsf{grow}$	1	1

$$P\left([_{\mathsf{S}}\ [_{\underset{+\mathsf{singular}}{\mathsf{NP}}}\ \mathsf{rice}]\ [_{\underset{+\mathsf{singular}}{\mathsf{VP}}}\ \mathsf{grows}]] \right) = 2/3$$

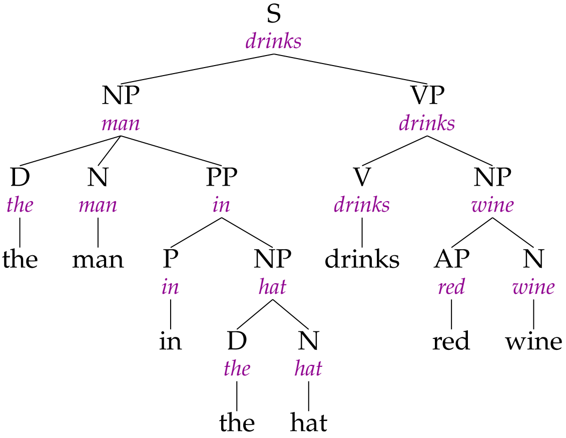
$$P\left([_{\mathsf{S}}\ [_{\underset{+\mathsf{plural}}{\mathsf{NP}}}\ \mathsf{bananas}]\ [_{\underset{+\mathsf{plural}}{\mathsf{VP}}}\ \mathsf{grow}]] \right) = 1/3$$

Outline

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
- These dependencies are encoded as "features" on nodes
- Most combinations of features are not observed in training data, but will occur in new sentences
  $\Rightarrow$ smoothing is essential!

“Head to head” dependencies



Rules:

$$\underset{\textit{drinks}}{\mathrm{S}} \rightarrow \underset{\textit{man}}{\mathrm{NP}}\ \underset{\textit{drinks}}{\mathrm{VP}}$$

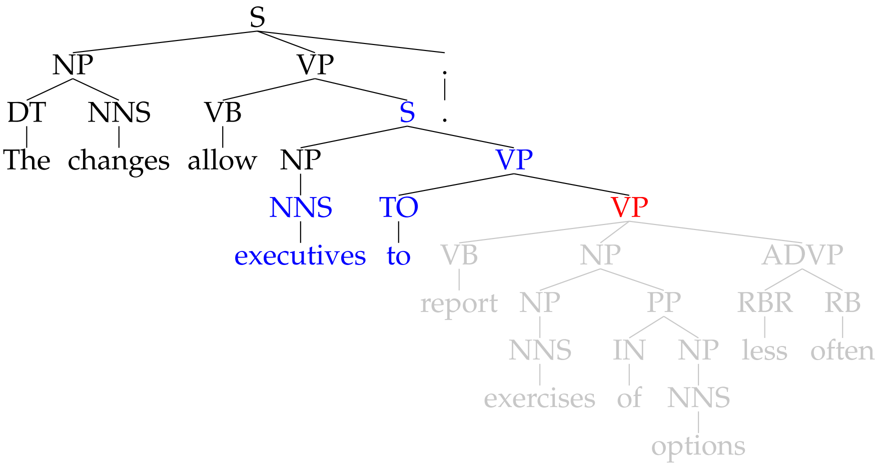
$$\underset{\textit{drinks}}{\mathrm{VP}} \rightarrow \underset{\textit{drinks}}{\mathrm{V}}\ \underset{\textit{wine}}{\mathrm{NP}}$$

$$\underset{\textit{wine}}{\mathrm{NP}} \rightarrow \underset{\textit{red}}{\mathrm{AP}}\ \underset{\textit{wine}}{\mathrm{N}}$$

...

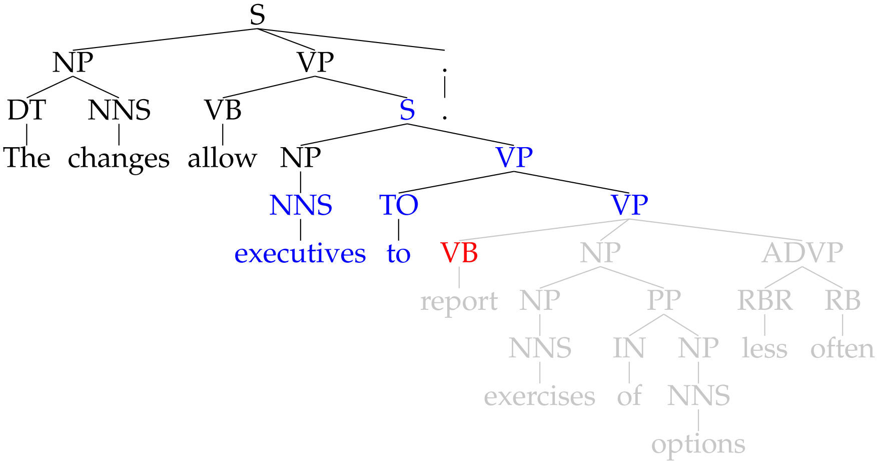
- *Lexicalization* captures syntactic and semantic dependencies
- Lexicalized structural preferences may be most important

Generative language model (Charniak 2001)



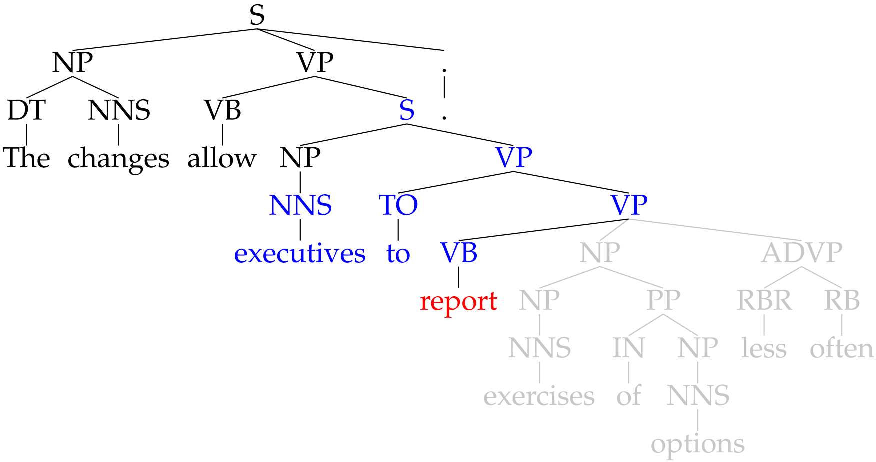
- Predicted node is shown in red
- Conditioning nodes are shown in blue

Generative language model (Charniak 2001)



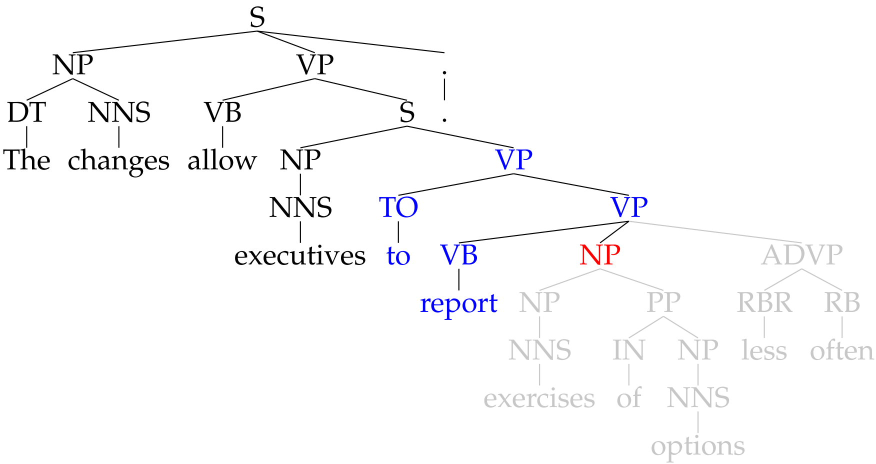
- Predicted node is shown in red
- Conditioning nodes are shown in blue

Generative language model (Charniak 2001)



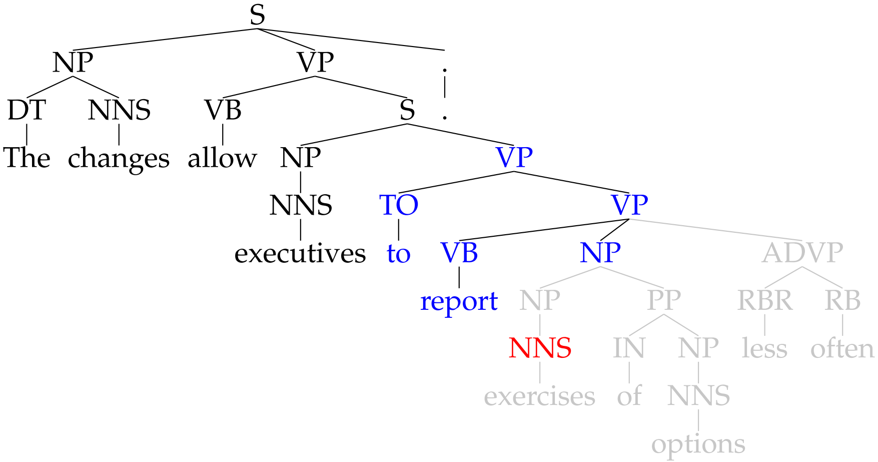
- Predicted node is shown in red
- Conditioning nodes are shown in blue

Generative language model (Charniak 2001)



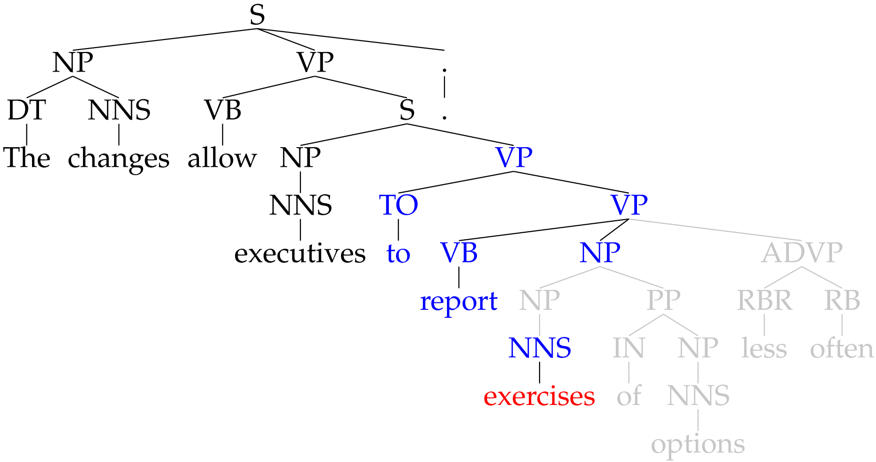
- Predicted node is shown in red
- Conditioning nodes are shown in blue

Generative language model (Charniak 2001)



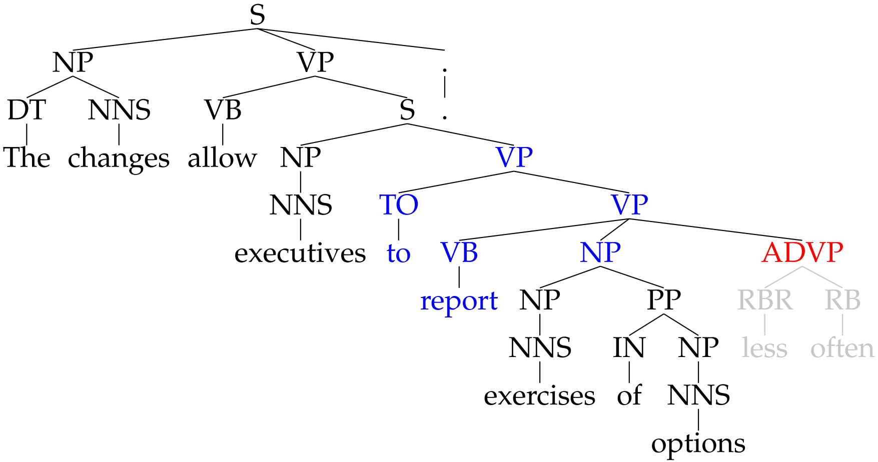
- Predicted node is shown in red
- Conditioning nodes are shown in blue

Generative language model (Charniak 2001)



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

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

$\Rightarrow$ relative frequency estimator not MLE

  - Make non-local dependencies local by splitting categories

  $\Rightarrow$ Astronomical number of possible categories

- Find some way of accurately estimating models in the presence of unmodeled dependencies

  $\Rightarrow$ exponential models

Outline

Exponential models

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

Let $f_1, \ldots, f_m$ be features, and $\lambda_1, \ldots, \lambda_m$ be real-valued feature weights. An *exponential model* has the form:

$$
\begin{aligned}
\mathrm{P}_\lambda(\psi) &= \frac{W_\lambda(\psi)}{Z_\lambda} \\
W_\lambda(\psi) &= \exp \sum_{j=1}^{m} \lambda_j f_j(\psi) \\
Z_\lambda &= \sum_{\psi' \in \Psi_G} W_\lambda(\psi')
\end{aligned}
$$

$W_\lambda(\psi)$ is the weight (unnormalized probability) of parse ψ.

Z_λ is called the *partition function*.

Exponential models are also known as *Gibbs models*, *log-linear models* and *Maximum Entropy models*.

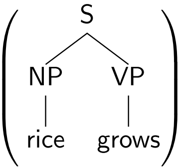
PCFGs are exponential models

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 $\lambda_j = \log p(r_j)$

$\boldsymbol{f}$ 

$$= [\underbrace{1}_{\text{S}\to\text{NP VP}}, \underbrace{1}_{\text{NP}\to\text{rice}}, \underbrace{0}_{\text{NP}\to\text{bananas}}, \underbrace{1}_{\text{VP}\to\text{grows}}, \underbrace{0}_{\text{VP}\to\text{grow}}]$$

$$\mathrm{P}(\psi) = \prod_{j=1}^{m} p(r_j)^{f_j(\psi)} = \prod_{j=1}^{m} (\exp \lambda_j)^{f_j(\psi)} = \exp \sum_{j=1}^{m} \lambda_j f_j(\psi)$$

So *a PCFG is just an exponential model* with $Z_\lambda = 1$.

Advantages of exponential models

- Exponential models are very flexible …
- Features f can be *any function of parses* …
  - whether a particular structure occurs in a parse
  - conjunctions of prosodic and syntactic structure
- Parses ψ need not be trees, but *can be anything at all*
  - Feature structures (LFG, HPSG)
- Exponential models are related to other popular models
  - Harmony theory and optimality theory
  - They are also called *Maximum Entropy* models and *log-linear* models

Modeling dependencies

- It's usually difficult to design a PCFG model that captures a particular set of dependencies
  - probability of the tree must be broken down into a product of *conditional probability distributions*
  - non-local dependencies must be expressed in terms of GPSG-style feature passing
- It's easy to make exponential models sensitive to new dependencies
  - add a new feature functions to existing feature functions
  - *figuring out what the right dependencies are is hard, but incorporating them into an exponential model is easy*

MLE of exponential models from visible data

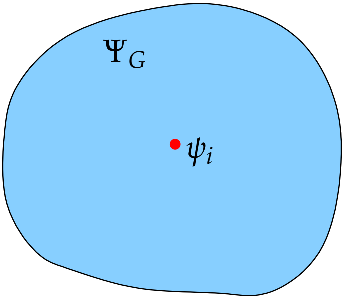
Visible training data: Parses $\Psi = \psi_1, \ldots, \psi_n$

$$\begin{aligned}
\log L(\boldsymbol{\lambda}) &= \sum_{i=1}^{n} \log \mathrm{P}_\lambda(\psi_i) \\
&= \sum_{i=1}^{n} \left(\log W_\lambda(\psi_i) - \log \sum_{\psi \in \Psi_G} W_\lambda(\psi) \right)
\end{aligned}$$

$$\frac{\partial \log L(\boldsymbol{\lambda})}{\partial \lambda_j} = \sum_{i=1}^{n} \left(f_j(\psi_i) - \mathrm{E}_\lambda[f_j] \right)$$

So the likelihood is maximized when the empirical frequency of each feature equals its expected frequency.

Maximizing likelihood of visible data is hard!



Maximizing likelihood requires summation over all of Ψ_G, even with fully visible data!

Maximizing likelihood contrasts the training data trees Ψ with Ψ_G; i.e., select λ to maximize $\sum_{i=1}^{n}(\log W_\lambda(\psi_i) - \log \sum_{\psi \in \Psi_G} W_\lambda(\psi))$.

But Ψ_G is the set of all parses of all sentences!

Estimation by maximizing conditional likelihood

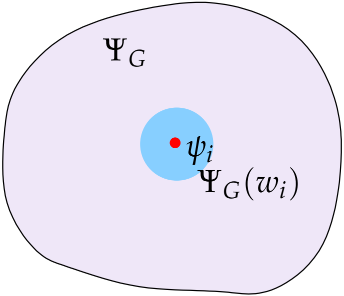
Maximize the *conditional likelihood* of the correct parses Ψ *given their yield* $\boldsymbol{w}$.

$$\begin{aligned}
\log L(\boldsymbol{\lambda}) &= \sum_{i=1}^{n} \log \mathrm{P}_{\lambda}(\psi_i | w_i) \\
&= \sum_{i=1}^{n} \left(\log W_{\lambda}(\psi_i) - \log \sum_{\psi \in \Psi_G(w_i)} W_{\lambda}(\psi) \right)
\end{aligned}$$

$$\frac{\partial \log L(\boldsymbol{\lambda})}{\partial \lambda_j} = \sum_{i=1}^{n} \left(f_j(\psi_i) - \mathrm{E}_{\lambda}[f_j | w_i] \right)$$

So conditional likelihood is maximized when the empirical frequency of each feature equals its expected frequency *conditioned on the yields*.

Maximizing conditional likelihood is easier



Pseudo-likelihood is consistent *for the conditional distribution*

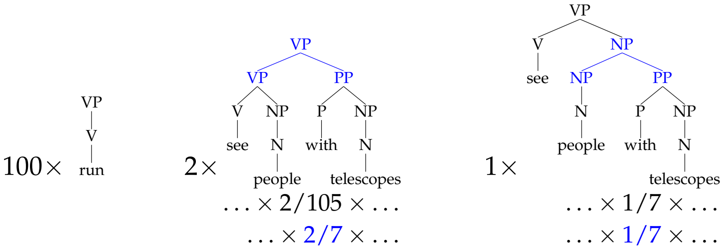
Maximizing conditional likelihood requires summing over $\Psi_G(w_i), i = 1, \ldots, n$ (obtained by parsing).

Conditional likelihood contrasts each element of training data ψ_i with the parses of w_i; i.e., adjust λ to maximize $\sum_{i=1}^{n}(\log W_\lambda(\psi_i) - \log \sum_{\psi' \in \Psi_G(w_i)} W_\lambda(\psi'))$.

Conditional likelihood is better for parsing

Parsing exploits $\mathrm{P}(\psi|w)$, which MCL optimizes.

If the grammar does not generate strings accurately, ML and MCL can be quite different!



Rule	**count**	**rel freq**	
VP → V	100	100/105	4/7
VP → V NP	3	3/105	1/7
VP → VP PP	2	2/105	**2/7**
NP → N	6	6/7	6/7
NP → NP PP	1	1/7	**1/7**

Conditional ML estimation

w_i	$\boldsymbol{f}(\psi_i)$	$\{\boldsymbol{f}(\psi) : \psi \in \Psi_G(w_i), \psi \neq \psi_i\}$
sentence 1	$(1,3,2)$	$(2,2,3)$ $(3,1,5)$ $(2,6,3)$
sentence 2	$(7,2,1)$	$(2,5,5)$
sentence 3	$(2,4,2)$	$(1,1,7)$ $(7,2,1)$
...	...	...

- Parser designer specifies *feature functions* $\boldsymbol{f} = (f_1, \ldots, f_m)$
- A parser produces trees $\Psi(w)$ for each sentence $w \in w_1, \ldots, w_n$
- Treebank tells us correct tree $\psi_i \in \Psi(w_i)$ for sentence w_i
- Feature functions $\boldsymbol{f}$ apply to each tree $\psi \in \Psi_G(w)$, producing feature values $\boldsymbol{f}(\psi) = (f_1(\psi), \ldots, f_m(\psi))$
- MCLE estimates feature weights $\widehat{\boldsymbol{\lambda}}$ using a gradient-based numerical optimizer

Regularization

- With a large number of features, exponential models can over-fit the training data
- Regularization: add *bias* term to ensure $\widehat{\boldsymbol{\lambda}}$ is finite and small
- In following experiments, regularizer is a polynomial penalty term

$$
\begin{aligned}
\widehat{\boldsymbol{\lambda}} &= \operatorname*{argmax}_{\boldsymbol{\lambda}} \log \sum_{i=1}^{n} \mathrm{P}_{\lambda}(\psi_i|w_i) - c \sum_{j=1}^{m} |\lambda_j|^p \\
&= \operatorname*{argmax}_{w} \sum_{i=1}^{n} \left(\sum_{j=1}^{m} \lambda_j f_j(\psi_i) - \log Z_{\lambda}(w_i) \right) - c \sum_{j=1}^{m} |\lambda_j|^p
\end{aligned}
$$

- $p = 2$ gives a *Gaussian prior*.
- We maximize this expression using *numerical optimization* (Limited Memory Variable Metric)

Conditional vs joint estimation

In this slide, let ψ be a parse tree without the terminal string w

$$\mathrm{P}(\psi, w) \; = \; \mathrm{P}(\psi|w)\mathrm{P}(w)$$

- ML optimizes probability of training trees ψ *and strings* w
- MCLE maximizes probability of trees given strings
  - Conditional estimation uses less information from the data
  - learns nothing from distribution of strings $\mathrm{P}(w)$
  - learns nothing from unambiguous sentences (!)
- Joint estimation should be better (lower variance) if your model correctly relates $\mathrm{P}(\psi|w)$ and $\mathrm{P}(w)$
- Conditional estimation should be better if your model incorrectly relates $\mathrm{P}(\psi|w)$ and $\mathrm{P}(w)$

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

Outline

Coarse to fine parsing

- Parsing with a grammar with a lot of features (PCFG nonterminals) is slow, even using the dynamic programming algorithms
- Coarse to fine parsing uses a sequence of grammars. The features of the coarse-grained grammars are equivalence classes of the fined-grained features.
- The parses produced by the coarse-grained grammars constrain the search with the fine-grained grammar.
- The Charniak generative parser uses a coarse-grained PCFG to identify which substrings should be parsed with the fine-grained PCFG.

Coarse to fine reranking with exponential models

- $Z_G(w)$ is still hard to compute $\Rightarrow$ make $\Psi(w)$ even smaller!
- Set $\Psi(w) =$ the 50-best parses produced by Charniak parser
- Exponential model is trained using MCLE to pick out best parse from Charniak's 50-best parses

$$
\begin{tikzcd}
 & w \arrow[dl] \arrow[dr] & & \text{Sentence} \\
 & & & \textit{Charniak parser} \\
\psi_1 \arrow[d] & \dots & \psi_{50} \arrow[d] & \text{Trees} \\
\boldsymbol{f}(\psi_1) \arrow[d] & \dots & \boldsymbol{f}(\psi_{50}) \arrow[d] & \text{Features} \\
\boldsymbol{\lambda} \cdot \boldsymbol{f}(\psi_1) & \dots & \boldsymbol{\lambda} \cdot \boldsymbol{f}(\psi_{50}) & \text{Parse scores}
\end{tikzcd}
$$

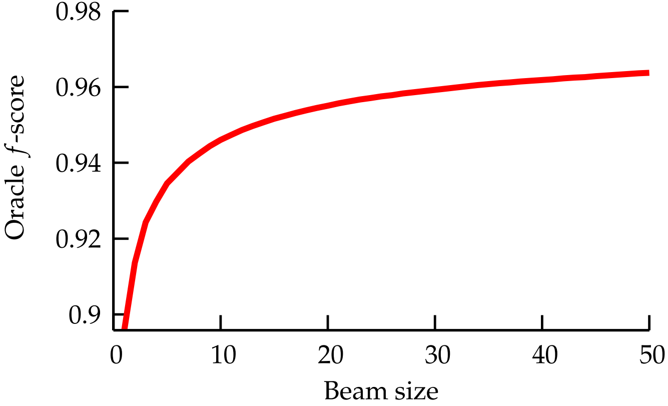
Features for ranking parses

- Features can be any real-valued function of parse trees
- In these experiments the features come in two kinds:
  - The logarithm of the tree's probability estimated by the Charniak parser
  - The number of times a particular configuration appears in the parse
- *Which ones improve parsing accuracy the most?* (can you guess?)

Experimental setup

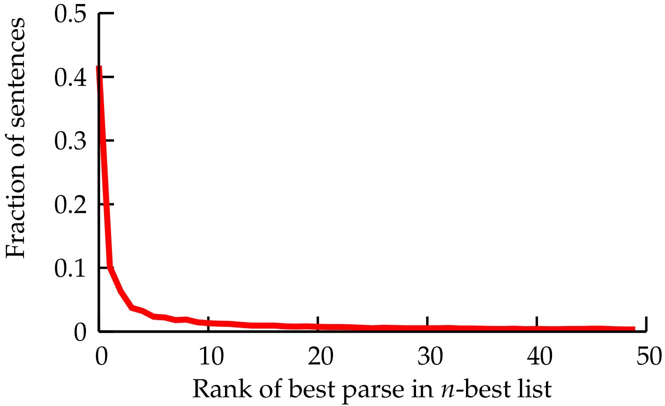
- Feature tuning experiments done using Collins' split: sections 2-19 as train, 20-21 as dev and 22 as test
- $\Psi(w)$ computed using Charniak 50-best parser
- Features which vary on less than 5 sentences pruned
- Optimization performed using LMVM optimizer from Petsc/TAO optimization package
- Regularizer constant c adjusted to maximize f-score on dev

f-score vs. n-best beam size



- F-score of Charniak's most probable parse = 0.896
- Oracle f-score (f-score of best parse in beam) of Charniak's 50-best parses = 0.965 (66% redn)

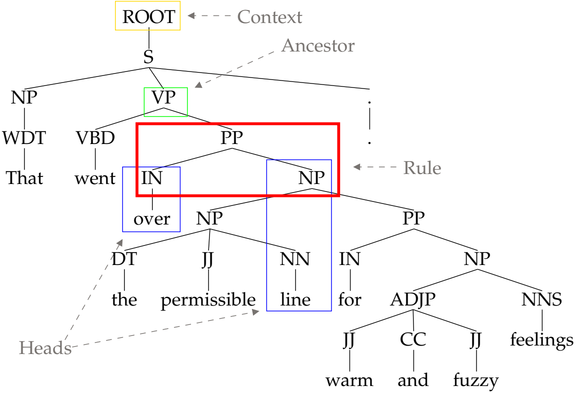
Rank of best parse



- Charniak parser's most likely parse is the best parse 41% of the time
- Reranker picks Charniak parser's most likely parse 58% of the time

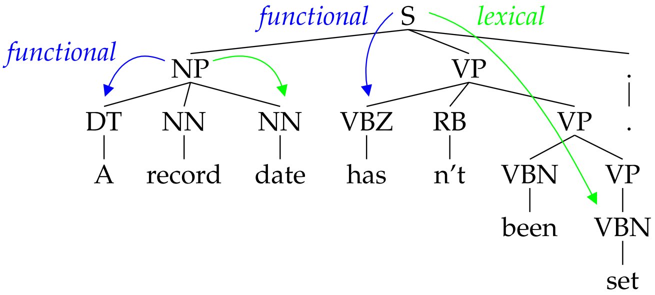
Lexicalized and parent-annotated rules

- Rule features largely replicate features already in generative parser
- A typical Rule feature might be (PP IN NP)



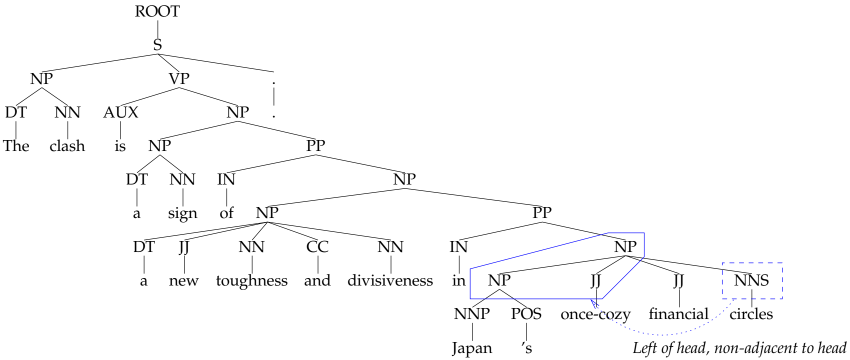
Functional and lexical heads

- There are at least two sensible notions of head (c.f., Grimshaw)
  - *Functional heads:* determiners of NPs, auxilary verbs of VPs, etc.
  - *Lexical heads:* rightmost Ns of NPs, main verbs in VPs, etc.
- In a log-linear model, it is easy to use both!



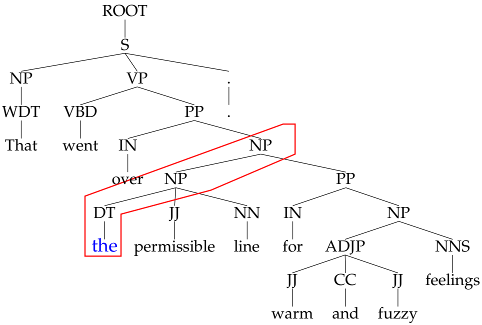
n-gram rule features generalize rules

- Breaks up long treebank constituents into shorter (phrase-like?) chunks
- Also includes *relationship to head* (e.g., adjacent? left or right?)



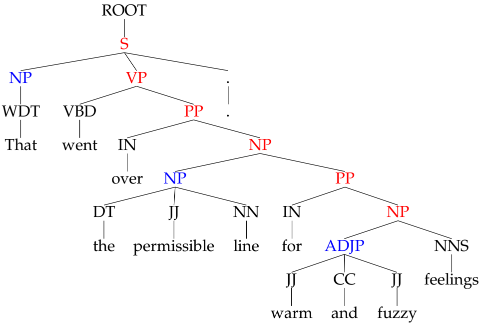
Word and WProj features

- 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



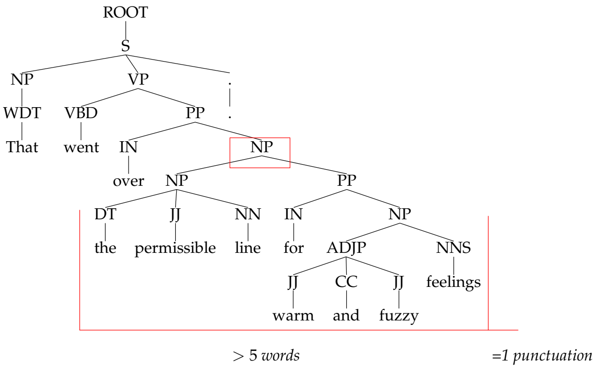
Rightmost branch bias

- 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!



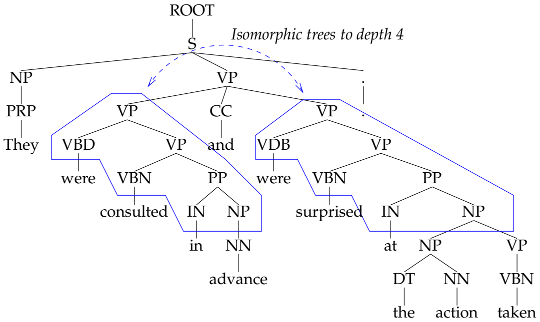
Constituent Heavyness and location

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



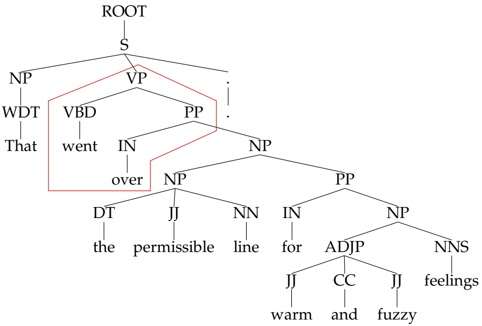
Coordination parallelism

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



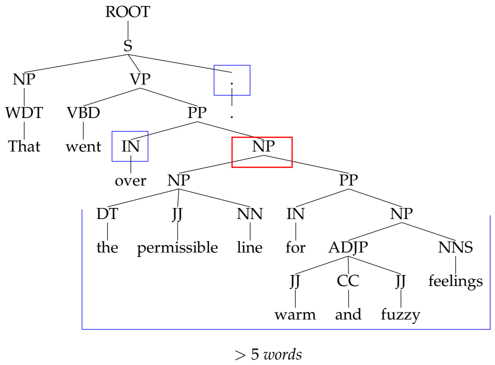
Tree n-gram

- 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

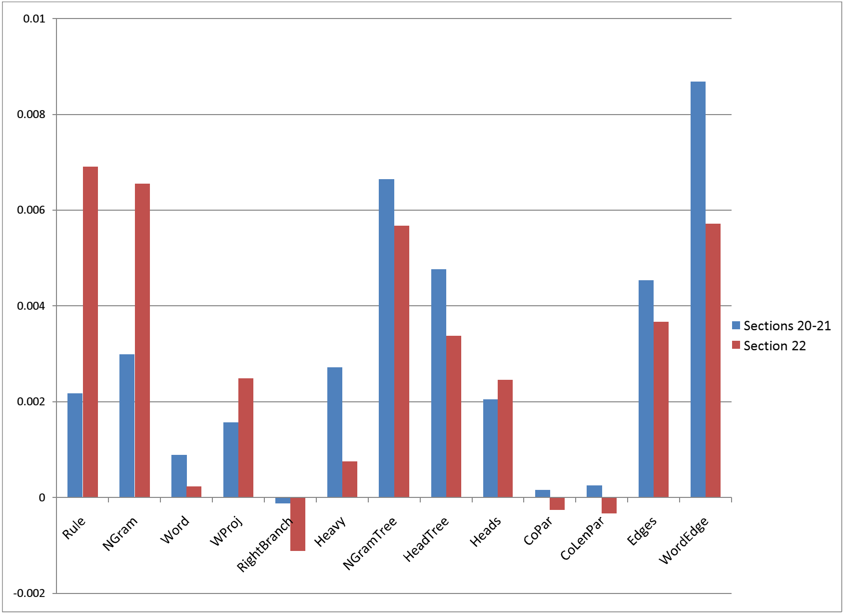


Edges and WordEdges

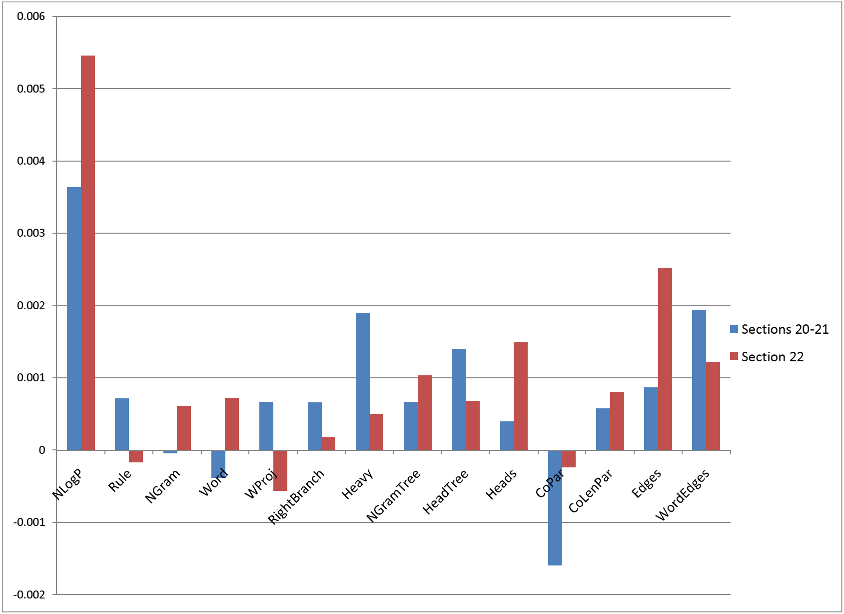
- A Neighbours feature indicates the node's category, its binned length and j left and k right lexical items and/or POS tags for $j, k \leq 2$



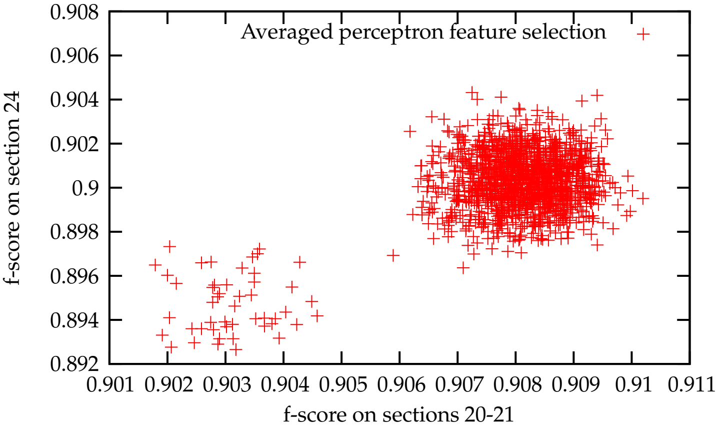
Adding one feature class to baseline parser



Removing one feature class from reranker



Feature selection is hard



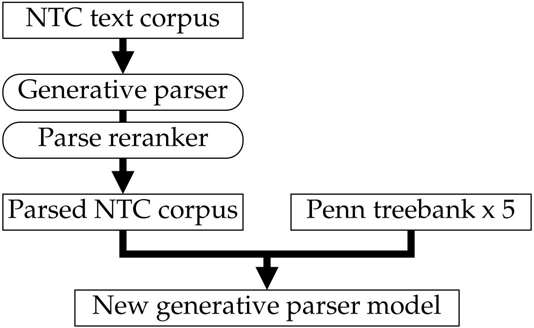
- Greedy feature selection using *averaged perceptron* optimizing f-score on sec 20–21
- All models also evaluated on section 24

Results on all training data

- Features must vary on parses of at least 5 sentences in training data
- In this experiment, 1,333,863 features
- Exponential model trained on sections 2-21
- Gaussian regularization $p = 2$, constant selected to optimize f-score on section 22
- On section 23: *recall = 91.0, precision = 91.8, f-score = 91.4*
- Available from www.cog.brown.edu

Outline

Self-training for reranking parsing



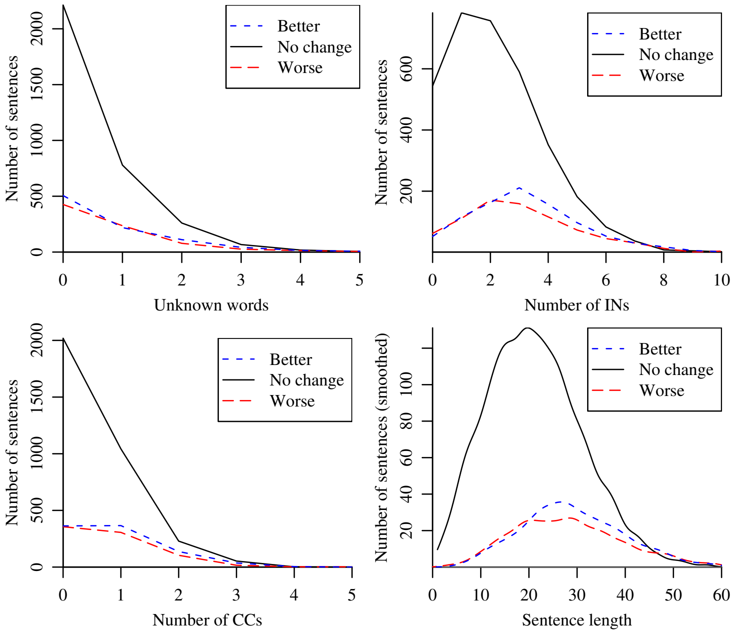
- *Improves performance* from 91.3 to **92.1 f-score**
- Self-training without the reranker does not improve performance
- Retraining the reranker on new first-stage model does not further improve performance
- Would reparsing the NTC with improved parser further improve performance?

First-stage oracle scores

Model	1-best	10-best	50-best
Baseline	89.0	94.0	95.9
WSJ×1 + 250k	89.8	94.6	96.2
WSJ×5 + 1,750k	90.4	94.8	96.4

- Self-training improves first-stage generative parser's oracle scores
- First-stage parser also became more decisive: mean of $\log_2$(P(1-best) / P(50th-best)) increased from 11.959 for the baseline parser to 14.104 for self-trained parser

Which sentences improve?



Self-trained WSJ parser on Brown

Sentences added	Parser	WSJ-reranker
Baseline Brown	86.4	87.4
Baseline WSJ	83.9	85.8
WSJ+50k	84.8	86.6
WSJ+250k	85.7	87.2
WSJ+1,000k	86.2	87.3
WSJ+2,500k	86.4	87.7

- Adding NTC data greatly improves performance on Brown corpus (to a lesser extent on Switchboard)

Self-training vs in-domain training

First-stage	First stage alone	WSJ-reranker	Brown-reranker
WSJ	82.9	85.2	85.2
WSJ+NTC	87.1	87.8	87.9
Brown	86.7	88.2	88.4

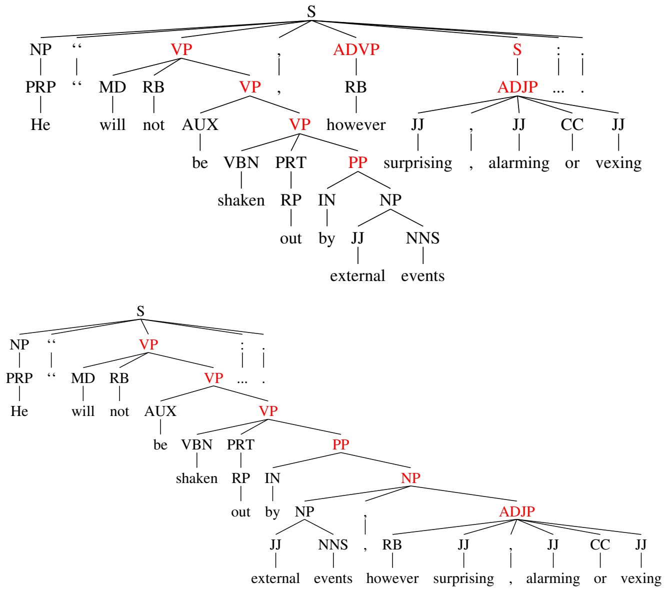
- Both reranking and self-training are surprisingly domain-independent
- Self-trained NTC parser with WSJ reranker is almost as good as a parser/reranker completely trained on Brown (!)

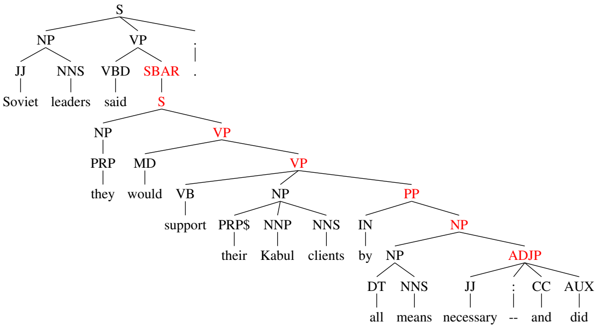
Summary and conclusions

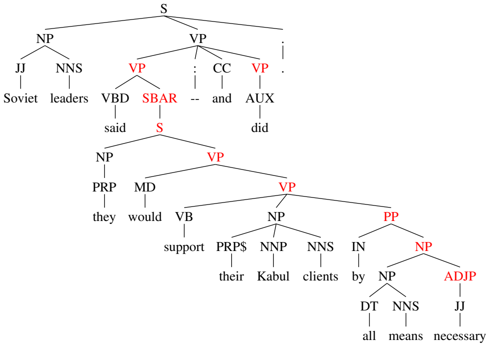
- PCFG based parsers are easy to estimate, but sensitive to unmodeled dependencies
- Exponential models are difficult to estimate, but resilient to unmodeled dependencies
- Coarse to fine reranking combines both approaches
- (Re)ranking parsers can work with just about any features
- The details of linguistic representations don't matter so long as they are rich enough to compute your features from
- Self-training works with reranking parsers (why?)
- Both reranking and self-training is (surprisingly) domain-independent

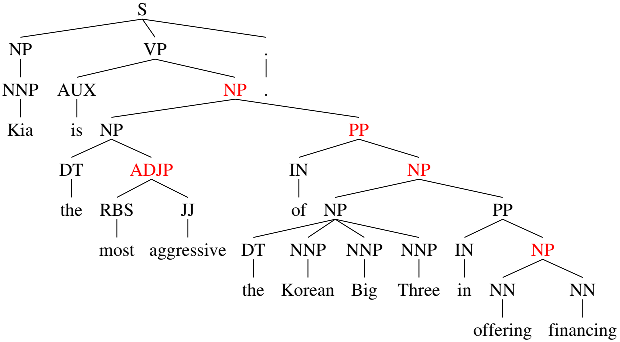
Outline

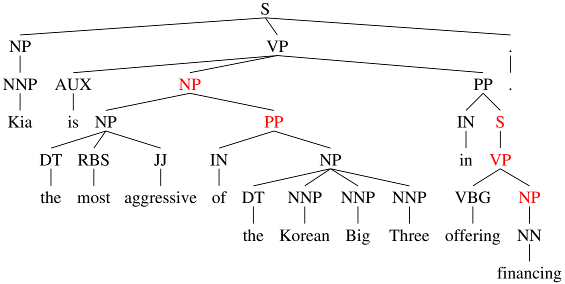
Sample parser errors

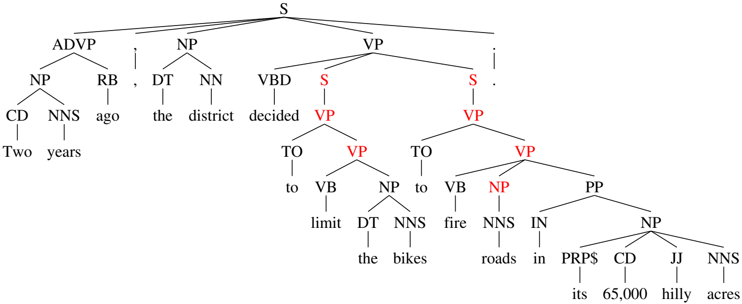


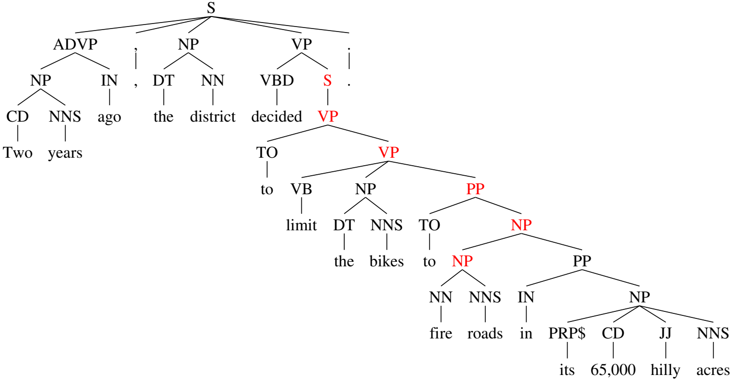

S
NP VP .
JJ NNS VBD SBAR .
Soviet leaders said S
NP VP
PRP MD VP
they would VB NP PP
support PRP$ NNP NNS IN NP
their Kabul clients by NP ADJP
DT NNS JJ : CC AUX
all means necessary -- and did

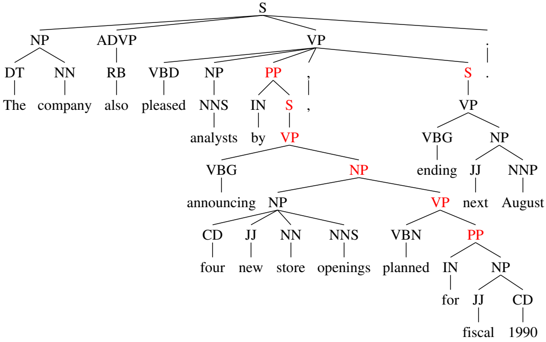

S
NP VP .
JJ NNS VP : CC VP .
Soviet leaders VBD SBAR -- and AUX
said S did
NP VP
PRP MD VP
they would VB NP PP
support PRP$ NNP NNS IN NP
their Kabul clients by NP ADJP
DT NNS JJ
all means necessary

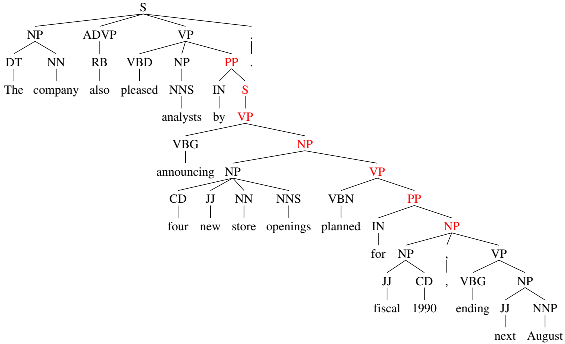

S
NP VP .
NNP AUX NP .
Kia is NP PP
DT ADJP IN NP
the RBS JJ of NP PP
most aggressive DT NNP NNP NNP IN NP
the Korean Big Three in NN NN
offering financing

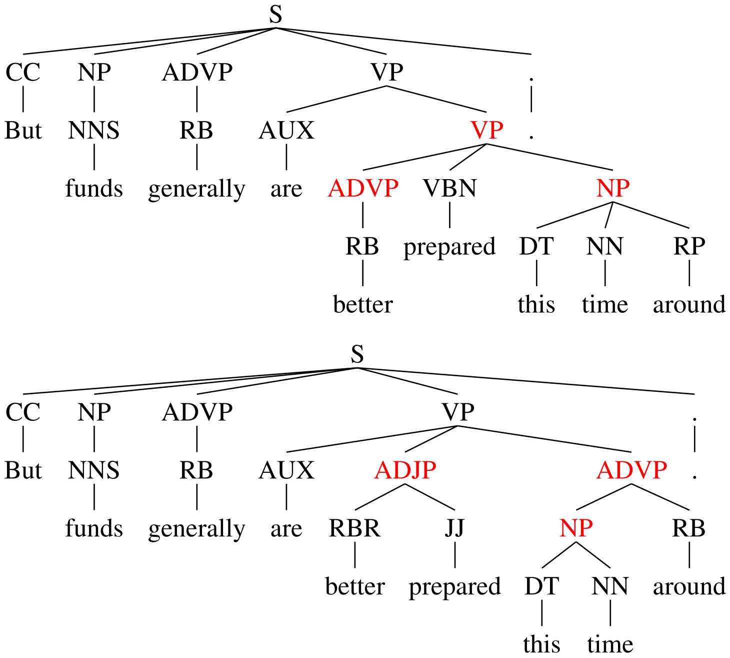

S
NP VP .
NNP AUX NP PP .
Kia is NP PP IN S
DT RBS JJ IN NP in VP
the most aggressive of DT NNP NNP NNP VBG NP
the Korean Big Three offering NN
financing


S
ADVP , NP VP .
NP RB , DT NN VBD S S .
CD NNS ago the district decided VP VP
Two years TO VP TO VP
to VB NP to VB NP PP
limit DT NNS fire NNS IN NP
the bikes roads in PRP$ CD JJ NNS
its 65,000 hilly acres


S
ADVP , NP VP .
NP IN , DT NN VBD S .
CD NNS ago the district decided VP
Two years TO VP
to VB NP PP
limit DT NNS TO NP
the bikes to NP PP
NN NNS IN NP
fire roads in PRP$ CD JJ NNS
its 65,000 hilly acres


S
NP ADVP VP .
DT NN RB VBD NP PP , S .
The company also pleased NNS IN S , VP
analysts by VP VBG NP
VBG NP ending JJ NNP
announcing NP VP next August
CD JJ NN NNS VBN PP
four new store openings planned IN NP
for JJ CD
fiscal 1990


S
NP ADVP VP .
DT NN RB VBD NP PP .
The company also pleased NNS IN S
analysts by VP
VBG NP
announcing NP VP
CD JJ NN NNS VBN PP
four new store openings planned IN NP
for NP , VP
JJ CD , VBG NP
fiscal 1990 ending JJ NNP
next August


S
CC NP ADVP VP .
But NNS RB AUX VP .
funds generally are ADVP VBN NP
RB prepared DT NN RP
better this time around
S
CC NP ADVP VP .
But NNS RB AUX ADJP ADVP .
funds generally are RBR JJ NP RB
better prepared DT NN around
this time