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 → 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.
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

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
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

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

- **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

\[
P \left( \begin{array}{c} S \\ NP & VP \\ rice & grows \end{array} \right) = 2/3
\]

\[
P \left( \begin{array}{c} S \\ NP & VP \\ corn & grows \end{array} \right) = 1/3
\]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Count</th>
<th>Rel Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NP → rice</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>NP → corn</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>VP → grows</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
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

\[
P \left( \begin{array}{c}
S \\
NP \\
\text{rice} \\
VP \\
grows
\end{array} \right) = \frac{4}{9}
\]

\[
P \left( \begin{array}{c}
S \\
NP \\
bananas \\
VP \\
grow
\end{array} \right) = \frac{1}{9}
\]

\[
\text{partition function } Z = \frac{5}{9}
\]
Dividing by partition function $Z$

$$P\left(\frac{S}{NP \mid VP} \right) = \frac{4}{9} \ 4/5$$

$$P\left(\frac{S}{NP \mid VP} \right) = \frac{1}{9} \ 1/5$$

$$Z = \frac{5}{9}$$
Other values do better!

\[
P \left( \begin{array}{c}
\text{S} \\
\text{NP} \\
\text{rice} \\
\text{VP} \\
grows
\end{array} \right) = \frac{2}{6} \quad \frac{2}{3}
\]

\[
P \left( \begin{array}{c}
\text{S} \\
\text{NP} \\
\text{bananas} \\
\text{VP} \\
grow
\end{array} \right) = \frac{1}{6} \quad \frac{1}{3}
\]

\[
Z = \frac{3}{6}
\]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Count</th>
<th>( p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>S \rightarrow NP VP</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NP \rightarrow rice</td>
<td>2</td>
<td>( \frac{2}{3} )</td>
</tr>
<tr>
<td>NP \rightarrow bananas</td>
<td>1</td>
<td>( \frac{1}{3} )</td>
</tr>
<tr>
<td>VP \rightarrow grows</td>
<td>2</td>
<td>( \frac{1}{2} )</td>
</tr>
<tr>
<td>VP \rightarrow grow</td>
<td>1</td>
<td>( \frac{1}{2} )</td>
</tr>
</tbody>
</table>

(Abney 1997)
Make dependencies local – GPSG-style

<table>
<thead>
<tr>
<th>rule</th>
<th>count</th>
<th>rel freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP + singular VP + singular</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>S → NP + plural VP + plural</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>NP + singular → rice</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>NP + plural → bananas</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>VP + singular → grows</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>VP + plural → grow</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
P \left( \begin{array}{ccc}
S & \text{NP} & \text{VP} \\
\text{NP} & \text{VP} & \\
+\text{singular} & +\text{singular} & \\
\text{rice} & \text{grows} & \\
\end{array} \right) = 2/3
\]

\[
P \left( \begin{array}{ccc}
S & \text{NP} & \text{VP} \\
\text{NP} & \text{VP} & \\
+\text{plural} & +\text{plural} & \\
\text{bananas} & \text{grow} & \\
\end{array} \right) = 1/3
\]
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
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 ⇒ smoothing is essential!
“Head to head” dependencies

Lexicalization captures syntactic and semantic dependencies

Lexicalized structural preferences may be most important
Generative language model (Charniak 2001)

The changes allow executives to report less often exercises of options.

- 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
The changes allow executives to report.

- 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
The changes allow executives to report exercises of options.

- 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.
  - 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.
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
Exponential models

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

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:

$$ P_\lambda(\psi) = \frac{W_\lambda(\psi)}{Z_\lambda} $$

$$ W_\lambda(\psi) = \exp \sum_{j=1}^{m} \lambda_j f_j(\psi) $$

$$ Z_\lambda = \sum_{\psi' \in \Psi_G} W_\lambda(\psi') $$

$W_\lambda(\psi)$ is the weight (unnormalized probability) of parse $\psi$. $Z_\lambda$ is called the *partition function*.

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

\[ \Psi = \text{set of all trees generated by PCFG } G \]
\[ f_j(\psi) = \text{number of times the } j\text{th rule is used in } \psi \]
\[ p(r_j) = \text{probability of } j\text{th rule in } G \]
Set weight \( \lambda_j = \log p(r_j) \)

\[ f \left( \begin{array}{c}
S \\
\text{NP} \\
rice \\
\text{VP} \\
grows
\end{array} \right) = \left[ \begin{array}{c}
1 \\
1 \\
0 \\
1 \\
0
\end{array} \right] \]
\[ \begin{array}{c}
\text{S} \rightarrow \text{NP} \text{ VP} \\
\text{NP} \rightarrow \text{rice} \\
\text{NP} \rightarrow \text{bananas} \\
\text{VP} \rightarrow \text{grows} \\
\text{VP} \rightarrow \text{grow}
\end{array} \]

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

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

$$\log L(\lambda) = \sum_{i=1}^{n} \log P_\lambda(\psi_i)$$

$$= \sum_{i=1}^{n} \left( \log W_\lambda(\psi_i) - \log \sum_{\psi \in \Psi_G} W_\lambda(\psi) \right)$$

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

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 $\Psi_G$, even with fully visible data!

Maximizing likelihood contrasts the training data trees $\Psi$ with $\Psi_G$; i.e., select $\lambda$ to maximize

$$\sum_{i=1}^{n} \left( \log W_\lambda(\psi_i) - \log \sum_{\psi \in \Psi_G} W_\lambda(\psi) \right).$$

But $\Psi_G$ is the set of all parses of all sentences!
Estimation by maximizing conditional likelihood

Maximize the conditional likelihood of the correct parses $\Psi$ given their yield $w$.

$$\log L(\lambda) = \sum_{i=1}^{n} \log P_{\lambda}(\psi_i|w_i)$$

$$= \sum_{i=1}^{n} \left( \log W_{\lambda}(\psi_i) - \log \sum_{\psi \in \Psi_G(w_i)} W_{\lambda}(\psi) \right)$$

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

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 $\psi_i$ with the parses of $w_i$; i.e., adjust $\lambda$ 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 $P(\psi|w)$, which MCL optimizes.

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

Rule | count | rel freq |
--- | --- | --- |
$VP \rightarrow V$ | 100 | $100/105$ | $4/7$ |
$VP \rightarrow V\ NP$ | 3 | $3/105$ | $1/7$ |
$VP \rightarrow VP\ PP$ | 2 | $2/105$ | $2/7$ |
$NP \rightarrow N$ | 6 | $6/7$ | $6/7$ |
$NP \rightarrow NP\ PP$ | 1 | $1/7$ | $1/7$
Conditional ML estimation

<table>
<thead>
<tr>
<th>$w_i$</th>
<th>$f(\psi_i)$</th>
<th>${f(\psi) : \psi \in \Psi_G(w_i), \psi \neq \psi_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence 1</td>
<td>(1, 3, 2)</td>
<td>(2, 2, 3) (3, 1, 5) (2, 6, 3)</td>
</tr>
<tr>
<td>sentence 2</td>
<td>(7, 2, 1)</td>
<td>(2, 5, 5)</td>
</tr>
<tr>
<td>sentence 3</td>
<td>(2, 4, 2)</td>
<td>(1, 1, 7) (7, 2, 1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Parser designer specifies feature functions $f = (f_1, \ldots, f_m)$
- 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 $f$ apply to each tree $\psi \in \Psi_G(w)$, producing feature values $f(\psi) = (f_1(\psi), \ldots, f_m(\psi))$
- MCLE estimates feature weights $\hat{\lambda}$ 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 \( \hat{\lambda} \) is finite and small.
- In following experiments, regularizer is a polynomial penalty term.

\[
\hat{\lambda} = \arg\max_{\lambda} \log \sum_{i=1}^{n} P_{\lambda}(\psi_i|\psi_i) - c \sum_{j=1}^{m} |\lambda_j|^p \\
= \arg\max_{\omega} \sum_{i=1}^{n} \left( \sum_{j=1}^{m} \lambda_j f_j(\psi_i) - \log Z_\lambda(\psi_i) \right) - c \sum_{j=1}^{m} |\lambda_j|^p
\]

- \( p = 2 \) gives a Gaussian prior.
- We maximize this expression using numerical optimization (Limited Memory Variable Metric).
Conditional vs joint estimation

In this slide, let $\psi$ be a parse tree without the terminal string $w$

$$P(\psi, w) = P(\psi|w)P(w)$$

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

- Probability of a parse $\psi$ is completely determined by its feature vector $(f_1(\psi), \ldots, f_m(\psi))$
- The actual linguistic representation of parse $\psi$ 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

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
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 fine-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
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)
There are at least two sensible notions of head (c.f., Grimshaw)

- **Functional heads:** determiners of NPs, auxiliary 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!
\textit{n}-gram rule features generalize rules

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

```
ROOT
  S
  NP
    DT NN AUX NP
      is NP PP
        IN NP
          IN NP
            IN NP
              NP
                IN NP
                  JJ CC NN
                    IN NP
                      IN NP
                        IN NP
                          IN NP
                            NP
                                PP
                                    NP
                                      PP
                                        NP
                                          NP
                                            NP
                                              NNS
```

\textit{Left of head, non-adjacent to head}
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 tendency 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.
• 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 connects sequences of adjacent $n$ words, for $n = 2, 3, 4$ (c.f. Bod’s DOP models)
- lexicalized and non-lexicalized variants

```
ROOT
  S
  NP
    WDT That
    VP
      VBD went
      IN over
    PP
      NP
        DT the
        JJ permissible
        NN line
      IN for
    NP
      ADJP warm
      CC and
      JJ fuzzy
      NNS feelings
```
• 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

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
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

<table>
<thead>
<tr>
<th>Model</th>
<th>1-best</th>
<th>10-best</th>
<th>50-best</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>89.0</td>
<td>94.0</td>
<td>95.9</td>
</tr>
<tr>
<td>WSJ×1 + 250k</td>
<td>89.8</td>
<td>94.6</td>
<td>96.2</td>
</tr>
<tr>
<td>WSJ×5 + 1,750k</td>
<td>90.4</td>
<td>94.8</td>
<td>96.4</td>
</tr>
</tbody>
</table>

- Self-training improves first-stage generative parser’s oracle scores
- First-stage parser also became more decisive: mean of log₂(\(P(1\text{-best}) / P(50\text{-th-best})\)) increased from 11.959 for the baseline parser to 14.104 for self-trained parser
Which sentences improve?

![Graphs showing the distribution of sentences across different types of changes: Unknown words, Number of INs, Number of CCs, and Sentence length. Each graph plots the number of sentences against the measure, with lines indicating Better, No change, and Worse scenarios.](image-url)
### Self-trained WSJ parser on Brown

<table>
<thead>
<tr>
<th>Sentences added</th>
<th>Parser</th>
<th>WSJ-reranker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Brown</td>
<td>86.4</td>
<td>87.4</td>
</tr>
<tr>
<td>Baseline WSJ</td>
<td>83.9</td>
<td>85.8</td>
</tr>
<tr>
<td>WSJ+50k</td>
<td>84.8</td>
<td>86.6</td>
</tr>
<tr>
<td>WSJ+250k</td>
<td>85.7</td>
<td>87.2</td>
</tr>
<tr>
<td>WSJ+1,000k</td>
<td>86.2</td>
<td>87.3</td>
</tr>
<tr>
<td>WSJ+2,500k</td>
<td>86.4</td>
<td>87.7</td>
</tr>
</tbody>
</table>

- Adding NTC data greatly improves performance on Brown corpus (to a lesser extent on Switchboard)
Self-training vs in-domain training

<table>
<thead>
<tr>
<th>First-stage</th>
<th>First stage alone</th>
<th>WSJ-reranker</th>
<th>Brown-reranker</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSJ</td>
<td>82.9</td>
<td>85.2</td>
<td>85.2</td>
</tr>
<tr>
<td>WSJ+NTC</td>
<td>87.1</td>
<td>87.8</td>
<td>87.9</td>
</tr>
<tr>
<td>Brown</td>
<td>86.7</td>
<td>88.2</td>
<td>88.4</td>
</tr>
</tbody>
</table>

- 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

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
Sample parser errors

He will not be shaken out by external events, however surprising, alarming or vexing.

S

NP ""

PRP ""

He

MD

RB

will

not

AUX

be

VBN

shaken

PRT

in

by

external

events

VP

,  

RB

however

JJ

JJ

CC

JJ

ADVP

S

ADJP ...
Soviet leaders said they would support their Kabul clients by all means necessary: -- and did.
Kia is the most aggressive of the Korean Big Three in offering financing.
Two years ago, the district decided to limit the bikes to fire roads in its 65,000 hilly acres.
The company also pleased analysts by announcing four new store openings planned for fiscal 1990, ending next August.
But funds generally are better prepared this time around.