Log-Linear Models with Structured Outputs (continued)

Introduction to Natural Language Processing
Computer Science 585—Fall 2009
University of Massachusetts Amherst

David Smith
Overview

- What computations do we need?
- Smoothing log-linear models
- MEMMs vs. CRFs again
  - Action-based parsing and dependency parsing
Recipe for Conditional Training of $p(y \mid x)$

1. Gather constraints/features from training data
   \[ \alpha_{iy} = \tilde{E} f_{iy} = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j) \]

2. Initialize all parameters to zero

3. Classify training data with current parameters; calculate expectations
   \[ E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y' \mid x_j) f_{iy}(x_j, y') \]

4. Gradient is
   \[ \tilde{E}[f_{iy}] - E_\Theta[f_{iy}] \]

5. Take a step in the direction of the gradient

6. Repeat from 3 until convergence
Recipe for Conditional Training of \( p(y \mid x) \)

1. Gather constraints/features from training data
\[
\alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)
\]

2. Initialize all parameters to zero

3. Classify training data with current parameters; calculate expectations
\[
E_{\Theta}[f_{iy}] = \sum_{x_j \in D} \sum_{y' \in y} p_{\Theta}(y' \mid x_j) f_{iy}(x_j, y')
\]

4. Gradient is
\[
\tilde{E}[f_{iy}] - E_{\Theta}[f_{iy}]
\]

5. Take a step in the direction of the gradient

6. Repeat from 3 until convergence

Where have we seen expected counts before?
Recipe for Conditional Training of \( p(y \mid x) \)

1. Gather constraints/features from training data
   \[
   \alpha_{iy} = \tilde{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)
   \]

2. Initialize all parameters to zero

3. Classify training data with current parameters; calculate expectations
   \[
   E_\Theta[f_{iy}] = \sum_{x_j \in D} \sum_{y'} p_\Theta(y' \mid x_j) f_{iy}(x_j, y')
   \]

4. Gradient is
   \[
   \tilde{E}[f_{iy}] - E_\Theta[f_{iy}]
   \]

5. Take a step in the direction of the gradient

6. Repeat from 3 until convergence

Where have we seen expected counts before? EM!
Gradient-Based Training

- $\lambda \leftarrow \lambda + \text{rate} \times \text{Gradient}(F)$
- After all training examples? (batch)
- After every example? (on-line)
- Use second derivative?
- A big field: numerical optimization
Overfitting

• If we have too many features, we can choose weights to model the training data perfectly

• If we have a feature that only appears in spam training, not ham training, it will get weight $\infty$ to maximize $p(\text{spam} \mid \text{feature})$ at 1.

• These behaviors
  • Overfit the training data
  • Will probably do poorly on test data
Solutions to Overfitting

- Throw out rare features.
  - Require every feature to occur > 4 times, and > 0 times with
    ling, and > 0 times with spam.
- Only keep, e.g., 1000 features.
  - Add one at a time, always greedily picking the one that most
    improves performance on held-out data.
- Smooth the observed feature counts.
- Smooth the weights by using a prior.
  - $\max p(\lambda|\text{data}) = \max p(\lambda, \text{data}) = p(\lambda)p(\text{data}|\lambda)$
  - decree $p(\lambda)$ to be high when most weights close to 0
Smoothing with Priors

• What if we had a prior expectation that parameter values wouldn’t be very large?

• We could then balance evidence suggesting large (or infinite) parameters against our prior expectation.

• The evidence would never totally defeat the prior, and parameters would be smoothed (and kept finite)

• We can do this explicitly by changing the optimization objective to maximum posterior likelihood:

$$\log P(y, \lambda | x) = \log P(\lambda) + \log P(y | x, \lambda)$$

<table>
<thead>
<tr>
<th>Posterior</th>
<th>Prior</th>
<th>Likelihood</th>
</tr>
</thead>
</table>
Smoothing: Priors

- Gaussian, or quadratic, priors:
  - Intuition: parameters shouldn’t be large.
  - Formalization: prior expectation that each parameter will be distributed according to a gaussian with mean $\mu$ and variance $\sigma^2$.

$$P(\lambda_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(\lambda_i - \mu_i)^2}{2\sigma_i^2}\right)$$

- Penalizes parameters for drifting to far from their mean prior value (usually $\mu=0$).
- $2\sigma^2=1$ works surprisingly well.
Parsing as Structured Prediction
<table>
<thead>
<tr>
<th>Stack</th>
<th>Input remaining</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Book that flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Book)</td>
<td>that flight</td>
<td>reduce, Verb → book, (Choice #1 of 2)</td>
</tr>
<tr>
<td>(Verb)</td>
<td>that flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Verb that)</td>
<td>flight</td>
<td>reduce, Det → that</td>
</tr>
<tr>
<td>(Verb Det)</td>
<td>flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Verb Det flight)</td>
<td></td>
<td>reduce, Noun → flight</td>
</tr>
<tr>
<td>(Verb Det Noun)</td>
<td></td>
<td>reduce, NOM → Noun</td>
</tr>
<tr>
<td>(Verb Det NOM)</td>
<td></td>
<td>reduce, NP → Det NOM</td>
</tr>
<tr>
<td>(Verb NP)</td>
<td></td>
<td>reduce, VP → Verb NP</td>
</tr>
<tr>
<td>(Verb)</td>
<td></td>
<td>reduce, S → V</td>
</tr>
<tr>
<td>(S)</td>
<td></td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>

Ambiguity may lead to the need for backtracking.
### Shift-reduce parsing

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input remaining</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Book that flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Book)</td>
<td>that flight</td>
<td>reduce, Verb → book, (Choice #1 of 2)</td>
</tr>
<tr>
<td>(Verb)</td>
<td>that flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Verb that)</td>
<td>flight</td>
<td>reduce, Det → that</td>
</tr>
<tr>
<td>(Verb Det)</td>
<td>flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Verb Det flight)</td>
<td></td>
<td>reduce, Noun → flight</td>
</tr>
<tr>
<td>(Verb Det Noun)</td>
<td></td>
<td>reduce, NOM → Noun</td>
</tr>
<tr>
<td>(Verb Det NOM)</td>
<td></td>
<td>reduce, NP → Det NOM</td>
</tr>
<tr>
<td>(Verb NP)</td>
<td></td>
<td>reduce, VP → Verb NP</td>
</tr>
<tr>
<td>(Verb)</td>
<td></td>
<td>reduce, S → V</td>
</tr>
<tr>
<td>(S)</td>
<td></td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>

Ambiguity may lead to the need for backtracking.
Shift-reduce parsing

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input remaining</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Book that flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Book)</td>
<td>that flight</td>
<td>reduce, Verb → book, (Choice #1 of 2)</td>
</tr>
<tr>
<td>(Verb)</td>
<td>that flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Verb that)</td>
<td>flight</td>
<td>reduce, Det → that</td>
</tr>
<tr>
<td>(Verb Det)</td>
<td>flight</td>
<td>shift</td>
</tr>
<tr>
<td>(Verb Det flight)</td>
<td></td>
<td>reduce, Noun → flight</td>
</tr>
<tr>
<td>(Verb Det Noun)</td>
<td></td>
<td>reduce, NOM → Noun</td>
</tr>
<tr>
<td>(Verb Det NOM)</td>
<td></td>
<td>reduce, NP → Det NOM</td>
</tr>
<tr>
<td>(Verb NP)</td>
<td></td>
<td>reduce, VP → Verb NP</td>
</tr>
<tr>
<td>(Verb)</td>
<td></td>
<td>reduce, S → V</td>
</tr>
<tr>
<td>(S)</td>
<td></td>
<td>SUCCESS!</td>
</tr>
</tbody>
</table>

Ambiguity may lead to the need for backtracking.

Train log-linear model of p(action | context)
He reckons the current account deficit will narrow to only 1.8 billion in September.
He reckons the current account deficit will narrow to only 1.8 billion in September.

POS-tagged sentence

He reckons the current account deficit will narrow to only 1.8 billion in September.

PRP    VBZ    DT    JJ    NN    NN    MD    VB    TO    RB    CD    CD    IN    NNP    .
He reckons the current account deficit will narrow to only 1.8 billion in September.

Part-of-speech tagging

He reckons the current account deficit will narrow to only 1.8 billion in September.

Word dependency parsing

He reckons the current account deficit will narrow to only 1.8 billion in September.
He reckons the current account deficit will narrow to only 1.8 billion in September.

Part-of-speech tagging

POS-tagged sentence

He reckons the current account deficit will narrow to only 1.8 billion in September.

Word dependency parsing

Word dependency parsed sentence

He reckons the current account deficit will narrow to only 1.8 billion in September.

slide adapted from Yuji Matsumoto
He reckons the current account deficit will narrow to only 1.8 billion in September.
He reckons the current account deficit will narrow to only 1.8 billion in September.
He reckons the current account deficit will narrow to only 1.8 billion in September.
Word Dependency Parsing

Raw sentence
He reckons the current account deficit will narrow to only 1.8 billion in September.

POS-tagged sentence
He reckons the current account deficit will narrow to only 1.8 billion in September.
PRP VBZ DT JJ NN NN MD VB TO RB CD CD IN NNP .

Word dependency parsed sentence
He reckons the current account deficit will narrow to only 1.8 billion in September .

slide adapted from Yuji Matsumoto
He reckons the current account deficit will narrow to only 1.8 billion in September.

Part-of-speech tagging

PRP VBZ DT JJ NN NN MD VB TO RB CD CD IN NNP .

Word dependency parsing

He reckons the current account deficit will narrow to only 1.8 billion in September .
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

- In the beginning, we used generative models.

\[ p(A) \times p(B \mid A) \times p(C \mid A, B) \times p(D \mid A, B, C) \times \ldots \]
In the beginning, we used generative models.

\[ p(A) \times p(B \mid A) \times p(C \mid A,B) \times p(D \mid A,B,C) \times \ldots \]
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

- In the beginning, we used generative models.

\[ p(A) \times p(B \mid A) \times p(C \mid A', B) \times p(D \mid A', B, C) \times \ldots \]

each choice depends on a limited part of the history
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

- In the beginning, we used generative models.

\[ p(A) \cdot p(B \mid A) \cdot p(C \mid A, B) \cdot p(D \mid A, B, C) \cdot \ldots \]

each choice depends on a limited part of the history

but which dependencies to allow? \[ p(D \mid A, B, C) \]?

what if they're all worthwhile? \[ p(D \mid A, B, C) \]?

\[ \ldots p(D \mid A, B) \cdot p(C \mid A, B, D) \]?
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

\[ p(A) * p(B \mid A) * p(C \mid A, B) * p(D \mid A, B, C) * \ldots \]

which dependencies to allow? (given limited training data)
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

- In the beginning, we used generative models.

\[ p(A) \times p(B \mid A) \times p(C \mid A,B) \times p(D \mid A,B,C) \times \ldots \]

which dependencies to allow? (given limited training data)
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

- In the beginning, we used generative models.

\[ p(A) \times p(B \mid A) \times p(C \mid A, B) \times p(D \mid A, B, C) \times \ldots \]

which dependencies to allow? (given limited training data)

\[ \frac{1}{Z} \times \Phi(A) \times \Phi(B, A) \times \Phi(C, A) \times \Phi(C, B) \times \Phi(D, A, B) \times \Phi(D, B, C) \times \Phi(D, A, C) \times \ldots \]

throw them all in!
Great ideas in NLP: Log-linear models
(Berger, della Pietra, della Pietra 1996; Darroch & Ratcliff 1972)

- In the beginning, we used generative models.

\[
p(A) \times p(B \mid A) \times p(C \mid A, B) \times p(D \mid A, B, C) \times \ldots
\]

which dependences to allow? (given limited training data)

- Solution: Log-linear (max-entropy) modeling

\[
\frac{1}{Z} \times \Phi(A) \times \Phi(B, A) \times \Phi(C, A) \times \Phi(C, B) \times \Phi(D, A, B) \times \Phi(D, B, C) \times \Phi(D, A, C) \times \ldots
\]

throw them all in!

- Features may interact in arbitrary ways
- **Iterative scaling** keeps adjusting the feature weights until the model agrees with the training data.
How about structured outputs?
How about structured outputs?

- Log-linear models great for n-way classification
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences

```
v  a  n
```

find preferred tags
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences

But to allow fast dynamic programming, only use n-gram features
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences

```
find   preferred   tags
```

but to allow fast dynamic programming, only use n-gram features
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences

but to allow fast dynamic programming, only use n-gram features
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences

but to allow fast dynamic programming, only use n-gram features
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences
- Also good for dependency parsing

...find preferred links...
How about structured outputs?

- Log-linear models great for n-way classification
- Also good for predicting sequences

![Diagram](image)

- Also good for dependency parsing

![Diagram](image)
How about structured outputs?

...find preferred links...

but to allow fast dynamic programming or MST parsing, only use **single-edge** features
How about structured outputs?

...find preferred links...

but to allow fast dynamic programming or MST parsing, only use single-edge features
How about structured outputs?

...find preferred links...

but to allow fast dynamic programming or MST parsing, only use single-edge features
How about structured outputs?

...find preferred links...

but to allow fast dynamic programming or MST parsing, only use single-edge features
How about structured outputs?

...find preferred links...

but to allow fast dynamic programming or MST parsing, only use single-edge features
How about structured outputs?

...find preferred links...

but to allow fast dynamic programming or MST parsing, only use single-edge features
Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?

Byl jasný studený dubnový den a hodiny odbíjely třináctou

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?

"It was a bright cold day in April and the clocks were striking thirteen"
Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?

Byl jasný studený dubnový den a hodiny odbíjely třináctou

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?

jasný ← den
(“bright day”)

Byl jasný studený dubnový den a hodiny odbíjely třináctou

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

- Is this a good edge?

jasný ← den
("bright day")

“It was a bright cold day in April and the clocks were striking thirteen”
Is this a good edge?

jasný $\leftarrow$ den
(“bright day”)

jasný $\leftarrow$ N
(“bright NOUN”)

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

Is this a good edge?

jasný ← den
(“bright day”)

jasný ← N
(“bright NOUN”)

“...It was a bright cold day in April and the clocks were striking thirteen...”
Is this a good edge?

jasný ← den
(“bright day”)

jasný ← N
(“bright NOUN”)

A ← N

It was a bright cold day in April and the clocks were striking thirteen
Is this a good edge?

jasný ↔ den  
("bright day")

jasný ↔ N  
("bright NOUN")

“IT was a bright cold day in April and the clocks were striking thirteen”
Is this a good edge?

"It was a bright cold day in April and the clocks were striking thirteen"
Edge-Factored Parsers (McDonald et al. 2005)

- How about this competing edge?

“It was a bright cold day in April and the clocks were striking thirteen”
**Edge-Factored Parsers** (McDonald et al. 2005)

- How about this competing edge?

  "It was a bright cold day in April and the clocks were striking thirteen"
How about this competing edge?

“It was a bright cold day in April and the clocks were striking thirteen”
How about this competing edge?

jasný ← hodiny
("bright clocks")

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

- How about this competing edge?

jasný ← hodiny
("bright clocks")

... undertrained ...

"It was a bright cold day in April and the clocks were striking thirteen"
How about this competing edge?

jasný ← hodiny
("bright clocks")

... undertrained ...

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

How about this competing edge?

jasný ← hodiny
(“bright clocks”)
... undertrained ...

jasn ← hodi
(“bright clock,” stems only)

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

How about this competing edge?

jasný ← hodiny
(“bright clocks”)
... undertrained ...

jasn ← hodi
(“bright clock,”
stems only)

Byl jasný studený dubnový den a hodiny odbíjely třináctou
V A A A N J N V C
byl jasn stud dubn den a hodi odbí třin

“It was a bright cold day in April and the clocks were striking thirteen”
How about this competing edge?

jasný ← hodiny
("bright clocks")

... undertrained ...

jasn ← hodi
("bright clock,"
stems only)

Aplural ← Nsingular

“It was a bright cold day in April and the clocks were striking thirteen”
edge-factored parsers (McDonald et al. 2005)

How about this competing edge?

jasný ← hodiny
("bright clocks")
... undertrained ...

jasn ← hodi
("bright clock,"
  stems only)

A_{plural} ← N_{singular}

"It was a bright cold day in April and the clocks were striking thirteen"
How about this competing edge?

jasný ← hodiny
A ← N
where N follows a conjunction

jasn ← hodi
(“bright clock,” stems only)

A_{plural} ← N_{singular}

Byl jasný studený dubnový den a hodiny odbíjely třináctou
V A A A N J N V C
byl jasn stud dubn den a hodi odbí třin

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

Which edge is better?
- “bright day” or “bright clocks”?

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

"It was a bright cold day in April and the clocks were striking thirteen"
Which edge is better?

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?

```
It was a bright cold day in April and the clocks were striking thirteen
```

Which edge is better?

```
our current weight vector
```

```
Byl jasný studený dubnový den a hodiny odbíjely třináctou

V A A A A N J N V C

byl jasn stud dubn den a hodi odbí třin
```

V A A A A N J N V C

byl jasn stud dubn den a hodi odbí třin
Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
- Score of an edge $e = \theta \cdot \text{features}(e)$

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
- Score of an edge $e = \theta \cdot \text{features}(e)$
- Standard algos $\Rightarrow$ valid parse with max total score

“... byl jasný studený dubnový den a hodiny odbíjely třináctou
byl jasn stud dubn den a hodi odbí třin

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers  (McDonald et al. 2005)

- Which edge is better?
- Score of an edge $e = \theta \cdot \text{features}(e)$
- Standard algos $\Rightarrow$ valid parse with max total score

our current weight vector
Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
- Score of an edge \( e = \theta \cdot \text{features}(e) \)
- Standard algos \( \rightarrow \) valid parse with max total score

*can’t have both*

(one parent per word)
**Edge-Factored Parsers** (McDonald et al. 2005)

- Which edge is better?
- Score of an edge $e = \theta \cdot \text{features}(e)$
- Standard algos $\Rightarrow$ **valid** parse with max **total** score

- can't have both
  (one parent per word)

- can't have both
  (no crossing links)
Edge-Factored Parsers  (McDonald et al. 2005)

- Which edge is better?
- Score of an edge $e = \theta \cdot \text{features}(e)$
- Standard algos $\Rightarrow$ valid parse with max total score

Can't have both
(one parent per word)

Can't have all three
(no cycles)

Can't have both
(no crossing links)
Edge-Factored Parsers (McDonald et al. 2005)

- Which edge is better?
- Score of an edge \( e = \theta \cdot \text{features}(e) \)
- Standard algos \( \Rightarrow \) valid parse with max total score

- Can't have both (one parent per word)
- Can't have both (no crossing links)
- Can't have all three (no cycles)

Thus, an edge may lose (or win) because of a consensus of other edges.
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming

The cat in the hat wore a stovepipe. ROOT
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming

The cat in the hat wore a stovepipe. ROOT

let's vertically stretch this graph drawing

The cat in the hat wore a stovepipe.
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming

The cat in the hat wore a stovepipe.

ROOT

let's vertically stretch this graph drawing

ROOT

each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming

The cat in the hat wore a stovepipe. ROOT

let's vertically stretch this graph drawing

each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$

The cat in the hat wore a stovepipe

each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case

Each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
    - to score “cat $\leftarrow$ wore” link, not enough to know this is NP
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
  - to score “cat $\leftarrow$ wore” link, not enough to know this is NP
  - must know it’s rooted at “cat”

```
The cat in the hat wore a stovepipe.
```

Each subtree is a linguistic constituent (here a noun phrase).
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
  - to score “cat $\leftarrow$ wore” link, not enough to know this is NP
  - must know it’s rooted at “cat”
  - so expand nonterminal set by $O(n)$: \{NP$_{\text{the}}$, NP$_{\text{cat}}$, NP$_{\text{hat}}$, ...\}

Each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
    - to score “cat ⇐ wore” link, not enough to know this is NP
    - must know it’s rooted at “cat”
    - so expand nonterminal set by $O(n)$: \{NP_{the}, NP_{cat}, NP_{hat}, ...\}
  - so CKY’s “grammar constant” is no longer constant 😞

Each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

The cat in the hat wore a stovepipe

ROOT

each subtree is a linguistic constituent (here a noun phrase)
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
  - Solution: Use a different decomposition (Eisner 1996)
    - Back to $O(n^3)$

Each subtree is a linguistic constituent (here a noun phrase)
Spans vs. constituents

Two kinds of substring.

» **Constituent** of the tree: links to the rest only through its headword (root).

> The cat in the hat wore a stovepipe. \textit{ROOT}

» **Span** of the tree: links to the rest only through its endwords.

> The cat in the hat \textit{wore} a stovepipe. \textit{ROOT}
Decomposing a tree into spans

The cat in the hat wore a stovepipe. ROOT

The cat + cat in the hat wore a stovepipe. ROOT

cat in the hat wore + wore a stovepipe. ROOT

cat in + in the hat wore

in the hat + hat wore
Finding Highest-Scoring Parse
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
  - Solution: Use a different decomposition (Eisner 1996)
    - Back to $O(n^3)$
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
  - Solution: Use a different decomposition (Eisner 1996)
    - Back to $O(n^3)$
- Can play usual tricks for dynamic programming parsing
  - Further refining the constituents or spans
    - Allow prob. model to keep track of even more internal information
  - A*, best-first, coarse-to-fine
  - Training by EM etc.
Finding Highest-Scoring Parse

- Convert to context-free grammar (CFG)
- Then use dynamic programming
  - CKY algorithm for CFG parsing is $O(n^3)$
  - Unfortunately, $O(n^5)$ in this case
  - Solution: Use a different decomposition (Eisner 1996)
    - Back to $O(n^3)$
- Can play usual tricks for dynamic programming parsing
  - Further refining the constituents or spans
    - Allow prob. model to keep track of even more internal information
  - A*, best-first, coarse-to-fine
  - Training by EM etc.

require “outside” probabilities of constituents, spans, or links
Hard Constraints on Valid Trees

- Score of an edge $e = \theta \cdot \text{features}(e)$
- Standard algos $\Rightarrow$ valid parse with max total score

- Can't have both (one parent per word)
- Can't have all three (no cycles)

Thus, an edge may lose (or win) because of a consensus of other edges.
Hard Constraints on Valid Trees

can't have both
(no crossing links)
Non-Projective Parses

can't have both
(no crossing links)

The “projectivity” restriction.
Do we really want it?
Non-Projective Parses

ROOT I’ll give a talk tomorrow on bootstrapping

can’t have both
(no crossing links)

The “projectivity” restriction. Do we really want it?
Non-Projective Parses

ROOT

I ’ll give a talk tomorrow on bootstrapping

can't have both
(no crossing links)

The “projectivity” restriction.
Do we really want it?
Non-Projective Parses

ROOT  I ‘ll give a talk tomorrow on bootstrapping

subtree rooted at “talk”
is a discontiguous noun phrase

can't have both
(no crossing links)

The “projectivity” restriction.
Do we really want it?
Non-Projective Parses

ROOT | I’ll give a talk tomorrow on bootstrapping

occasional non-projectivity in English
Non-Projective Parses

I 'll give a talk tomorrow on bootstrapping

occasional non-projectivity in English

ista meam norit gloria canitiem

frequent non-projectivity in Latin, etc.
Non-Projective Parses

That glory may-know my going-gray
(i.e., it shall last till I go gray)

frequent non-projectivity in Latin, etc.
Non-Projective Parses

occasional non-projectivity in English

That glory may-know my going-gray
(i.e., it shall last till I go gray)

frequent non-projectivity in Latin, etc.
Non-Projective Parses

occasional non-projectivity in English

That glory may-know my going-gray (i.e., it shall last till I go gray)

frequent non-projectivity in Latin, etc.
Non-Projective Parses

ROOT I ‘ll give a talk tomorrow on bootstrapping

occasional non-projectivity in English

ROOT ista meam norit gloria canitiem
that_{NOM} my_{ACC} may-know glory_{NOM} going-gray_{ACC}

That glory may-know my going-gray
(i.e., it shall last till I go gray)

frequent non-projectivity in Latin, etc.
Finding highest-scoring non-projective tree

- Consider the sentence “John saw Mary” (left).
- The Chu-Liu-Edmonds algorithm finds the maximum-weight spanning tree (right) – may be non-projective.
- Can be found in time $O(n^2)$.

Every node selects best parent
If cycles, contract them and repeat
Consider the sentence “John saw Mary” (left).

The Chu-Liu-Edmonds algorithm finds the maximum-weight spanning tree (right) – may be non-projective.

Can be found in time $O(n^2)$.

How about total weight $Z$ of all trees?

How about outside probabilities or gradients?

Can be found in time $O(n^3)$ by matrix determinants and inverses (Smith & Smith, 2007).
Graph Theory to the Rescue!

Tutte’s **Matrix-Tree Theorem** (1948)

The **determinant** of the Kirchoff (aka Laplacian) adjacency matrix of directed graph $G$ without row and column $r$ is equal to the **sum of scores of all directed spanning trees** of $G$ rooted at node $r$. 
Graph Theory to the Rescue!

Tutte’s **Matrix-Tree Theorem** (1948)

The **determinant** of the Kirchoff (aka Laplacian) adjacency matrix of directed graph $G$ without row and column $r$ is equal to the **sum of scores of all directed spanning trees** of $G$ rooted at node $r$.

Exactly the $Z$ we need!
Graph Theory to the Rescue!

Tutte’s Matrix-Tree Theorem (1948)

The determinant of the Kirchoff (aka Laplacian) adjacency matrix of directed graph $G$ without row and column $r$ is equal to the sum of scores of all directed spanning trees of $G$ rooted at node $r$.

$O(n^3)$ time!

Exactly the $Z$ we need!
Building the Kirchoff (Laplacian) Matrix

\[
\begin{bmatrix}
0 & -s(1,0) & -s(2,0) & L & -s(n,0) \\
0 & 0 & -s(2,1) & L & -s(n,1) \\
0 & -s(1,2) & 0 & L & -s(n,2) \\
M & M & M & O & M \\
0 & -s(1,n) & -s(2,n) & L & 0
\end{bmatrix}
\]

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant
Building the Kirchoff (Laplacian) Matrix

\[
\begin{bmatrix}
0 & -s(1,0) & -s(2,0) & L & -s(n,0) \\
0 & 0 & -s(2,1) & L & -s(n,1) \\
0 & -s(1,2) & 0 & L & -s(n,2) \\
M & M & M & O & M \\
0 & -s(1,n) & -s(2,n) & L & 0
\end{bmatrix}
\]

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant
Building the Kirchoff (Laplacian) Matrix

\[
\begin{bmatrix}
0 & -s(1,0) & -s(2,0) & L & -s(n,0) \\
0 & \sum_{j \neq 1} s(1,j) & -s(2,1) & L & -s(n,1) \\
0 & -s(1,2) & \sum_{j \neq 2} s(2,j) & \Lambda & -s(n,2) \\
M & M & M & M & O \\
0 & -s(1,n) & -s(2,n) & L & \sum_{j \neq n} s(n,j)
\end{bmatrix}
\]

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant
Building the Kirchoff (Laplacian) Matrix

\[
\begin{array}{c|c|c|c}
\sum_{j \neq 1} s(1, j) & -s(2,1) & L & -s(n,1) \\
-s(1,2) & \sum_{j \neq 2} s(2, j) & L & -s(n,2) \\
M & M & O & M \\
-s(1, n) & -s(2, n) & L & \sum_{j \neq n} s(n, j)
\end{array}
\]

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant
Building the Kirchoff (Laplacian) Matrix

\[
\begin{vmatrix}
\sum_{j \neq 1} s(1, j) & -s(2,1) & L & -s(n,1) \\
-s(1,2) & \sum_{j \neq 2} s(2, j) & L & -s(n,2) \\
M & M & O & M \\
-s(1, n) & -s(2, n) & L & \sum_{j \neq n} s(n, j)
\end{vmatrix}
\]

- Negate edge scores
- Sum columns (children)
- Strike root row/col.
- Take determinant

N.B.: This allows multiple children of root, but see Koo et al. 2007.
Why Should This Work?

Clear for 1x1 matrix; use induction

Chu-Liu-Edmonds analogy:
Every node selects best parent
If cycles, contract and recur

\[ K' = K \text{ with contracted edge } 1,2 \]
\[ K'' = K(\{1,2\} \mid \{1,2\}) \]
\[ |K| = s(1,2) |K'| + |K''| \]

Undirected case; special root cases for directed
Why Should This Work?

Clear for 1x1 matrix; use induction

\[
\begin{vmatrix}
\sum_{j \neq 1} s(1, j) & -s(2,1) & L & -s(n,1) \\
-s(1,2) & \sum_{j \neq 2} s(2, j) & \Lambda & -s(n,2) \\
M & M & O & M \\
-s(1,n) & -s(2,n) & L & \sum_{j \neq n} s(n, j)
\end{vmatrix}
\]

\[K' = K \text{ with contracted edge } 1,2\]
\[K'' = K(\{1,2\} \mid \{1,2\})\]
\[|K| = s(1,2)|K'| + |K''|\]

Chu-Liu-Edmonds analogy:
Every node selects best parent
If cycles, contract and recur

Undirected case; special root cases for directed