Log-Linear Models with Structured Outputs

Natural Language Processing
CS 6120—Spring 2013
Northeastern University

David Smith
(some slides from Andrew McCallum)
Overview

• Sequence labeling task (cf. POS tagging)
• Independent classifiers
• HMMs
• (Conditional) Maximum Entropy Markov Models
• Conditional Random Fields
• Beyond Sequence Labeling
Sequence Labeling

• Inputs: \( x = (x_1, \ldots, x_n) \)
• Labels: \( y = (y_1, \ldots, y_n) \)
• Typical goal: Given \( x \), predict \( y \)

• Example sequence labeling tasks
  – Part-of-speech tagging
  – Named-entity-recognition (NER)
    • Label people, places, organizations
NER Example:

Red Sox and Their Fans Let Loose

BOSTON, Oct. 30 — Jonathan Papelbon turned Boston’s World Series victory parade into a full-scale dance party Tuesday as the Red Sox put an exclamation point on the 2007 season.
First Solution: Maximum Entropy Classifier

- Conditional model $p(y|x)$.
  - Do not waste effort modeling $p(x)$, since $x$ is given at test time anyway.
  - Allows more complicated input features, since we do not need to model dependencies between them.

- Feature functions $f(x,y)$:
  - $f_1(x,y) = \{ \text{word is Boston \& } y=\text{Location} \}$
  - $f_2(x,y) = \{ \text{first letter capitalized \& } y=\text{Name} \}$
  - $f_3(x,y) = \{ x \text{ is an HTML link \& } y=\text{Location} \}$
First Solution: MaxEnt Classifier

• How should we choose a classifier?

• Principle of maximum entropy
  – We want a classifier that:
    • Matches feature constraints from training data.
    • Predictions maximize entropy.

• There is a unique, exponential family distribution that meets these criteria.
First Solution: MaxEnt Classifier

- Problem with using a maximum entropy classifier for sequence labeling:
- It makes decisions at each position independently!
Second Solution: HMM

\[ P(y, x) = \prod_{t} P(y_{t} \mid y_{t-1}) P(x \mid y_{t}) \]

- Defines a generative process.
- Can be viewed as a weighted finite state machine.
Second Solution: HMM

• How can represent we multiple features in an HMM?
  – Treat them as conditionally independent given the class label?
    • The example features we talked about are not independent.
  – Try to model a more complex generative process of the input features?
    • We may lose tractability (i.e. lose a dynamic programming for exact inference).
Second Solution: HMM

• Let’s use a conditional model instead.
Third Solution: MEMM

- Use a series of maximum entropy classifiers that know the previous label.
- Define a Viterbi algorithm for inference.

\[ P(y \mid x) = \prod_{t} P_{y_{t-1}}(y_t \mid x) \]
Third Solution: MEMM

• Combines the advantages of maximum entropy and HMM!
• But there is a problem…
Problem with MEMMs: Label Bias

- In some state space configurations, MEMMs essentially completely ignore the inputs.

- This is not a problem for HMMs, because the input sequence is generated by the model.

![Diagram of state transitions](image-url)
Fourth Solution: Conditional Random Field

- Conditionally-trained, undirected graphical model.
- For a standard linear-chain structure:

\[
P(y \mid x) = \prod_{t} \Psi_k(y_t, y_{t-1}, x)
\]

\[
\Psi_k(y_t, y_{t-1}, x) = \exp\left(\sum_k \lambda_k f(y_t, y_{t-1}, x)\right)
\]
Fourth Solution: CRF

- Have the advantages of MEMMs, but avoid the label bias problem.

- CRFs are globally normalized, whereas MEMMs are locally normalized.

- Widely used and applied. CRFs give state-the-art results in many domains.
Fourth Solution: CRF

- Have the advantages of MEMMs, but avoid the label bias problem.

- CRFs are globally normalized, whereas MEMMs are locally normalized.

- Widely used and applied, state-of-the-art results in natural language processing.

Remember, $Z$ is the normalization constant. How do we compute it?
CRF Applications

- Part-of-speech tagging
- Named entity recognition
- Document layout (e.g. table) classification
- Gene prediction
- Chinese word segmentation
- Morphological disambiguation
- Citation parsing
- Etc., etc.
NER as Sequence Tagging

The Phoenicians came from the Red Sea
NER as Sequence Tagging

The Phoenicians came from the Red Sea
NER as Sequence Tagging

The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea
NER as Sequence Tagging

The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea

NER as Sequence Tagging

Not capitalized
NER as Sequence Tagging

The Phoenicians came from the Red Sea
NER as Sequence Tagging

The Phoenicians came from the Red Sea

- B-E to right
- Not capitalized
- Tagged as “VB”
NER as Sequence Tagging

The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea
The Phoenicians came from the Red Sea
NER as Sequence Tagging

The Phoenicians came from the Red Sea

Word “sea” preceded by “the ADJ”

Hard constraint: I-L must follow B-L or I-L
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
   $$\alpha_{iy} = \hat{E}[f_{iy}] = \sum_{x_j, y_j \in D} f_{iy}(x_j, y_j)$$

2. Initialize all parameters to zero

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

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

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

- $\lambda := \lambda + \text{rate} \times \text{Gradient}(F)$
- After all training examples? (batch)
- After every example? (on-line)
- Use second derivative for faster learning?
- 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.
  • \[ \text{max } p(\lambda|\text{data}) = \text{max } p(\lambda, \text{data}) = p(\lambda)p(\text{data}|\lambda) \]
  • 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:

\[
\log P(y, \lambda \mid x) = \log P(\lambda) + \log P(y \mid x, \lambda)
\]

Posterior Prior Likelihood
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 too far from their mean prior value (usually $\mu=0$).
- $2\sigma^2=1$ works surprisingly well.
Parsing as Structured Prediction
## 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 $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ flight</td>
</tr>
<tr>
<td>(Verb Det Noun)</td>
<td></td>
<td>reduce, NOM $\rightarrow$ Noun</td>
</tr>
<tr>
<td>(Verb Det NOM)</td>
<td></td>
<td>reduce, NP $\rightarrow$ Det NOM</td>
</tr>
<tr>
<td>(Verb NP)</td>
<td></td>
<td>reduce, VP $\rightarrow$ Verb NP</td>
</tr>
<tr>
<td>(Verb)</td>
<td></td>
<td>reduce, S $\rightarrow$ 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 $p(\text{action} | \text{context})$
### 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 \( p(\text{action} \mid \text{context}) \)**

**Compare to an MEMM**
Structured Log-Linear Models
Structured Log-Linear Models

- Linear model for scoring structures

\[
\text{score}(out, in) = \theta \cdot \text{features}(out, in)
\]
Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing

\[
\text{score}(\text{out}, \text{in}) = \theta \cdot \text{features}(\text{out}, \text{in})
\]

\[
p(\text{out} \mid \text{in}) = \frac{1}{Z} e^{\text{score}(\text{out}, \text{in})}
Z = \sum_{\text{out}' \in \text{GEN}(\text{in})} e^{\text{score}(\text{out}', \text{in})}
\]
Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models

\[
\text{score}(\text{out}, \text{in}) = \theta \cdot \text{features}(\text{out}, \text{in})
\]

\[
p(\text{out} \mid \text{in}) = \frac{1}{Z} e^{\text{score}(\text{out}, \text{in})} \quad Z = \sum_{\text{out}' \in \text{GEN}(\text{in})} e^{\text{score}(\text{out}', \text{in})}
\]
Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models

\[
p(out \mid in) = \frac{1}{Z} e^{score(out, in)}
\]

Usually the bottleneck in NLP

\[
score(out, in) = \theta \cdot \text{features}(out, in)
\]

\[
Z = \sum_{out' \in GEN(in)} e^{score(out', in)}
\]
Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models
- Inference: sampling, variational methods, dynamic programming, local search, ...

\[
p(out \mid in) = \frac{1}{Z} e^{\text{score}(out, in)} \quad Z = \sum_{out' \in \text{GEN}(in)} e^{\text{score}(out', in)}
\]

Usually the bottleneck in NLP

\[
\text{score}(out, in) = \theta \cdot \text{features}(out, in)
\]
Structured Log-Linear Models

- Linear model for scoring structures
- Get a probability distribution by normalizing
  - Viz. logistic regression, Markov random fields, undirected graphical models
- Inference: sampling, variational methods, dynamic programming, local search, ...
- Training: maximum likelihood, minimum risk, etc.

\[
p(out \mid in) = \frac{1}{Z} e^{score(out,in)} \quad Z = \sum_{out' \in GEN(in)} e^{score(out',in)}
\]

Usually the bottleneck in NLP
Structured Log-Linear Models

With latent variables

• Several layers of linguistic structure
• Unknown correspondences
• Naturally handled by probabilistic framework
• Several inference setups, for example:

\[ p(out_1 \mid in) = \sum_{out_2, \text{alignment}} p(out_1, out_2, \text{alignment} \mid in) \]
Structured Log-Linear Models

With latent variables

• Several layers of linguistic structure
• Unknown correspondences
• Naturally handled by probabilistic framework
• Several inference setups, for example:

\[ p(out_1 \mid in) = \sum_{out_2, \text{alignment}} p(out_1, out_2, \text{alignment} \mid in) \]
Edge-Factored Parsers

- No global features of a parse (McDonald et al. 2005)
- Each feature is attached to some edge
- MST or CKY-like DP for fast $O(n^2)$ or $O(n^3)$ parsing

“IT was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

- 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

• Is this a good edge?

yes, lots of positive features ...

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

- Is this a good edge?

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

• 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

• Is this a good edge?

jasný ← den
(“bright day”)

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

• 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”
Edge-Factored Parsers

• 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”
Edge-Factored Parsers

- Is this a good edge?

```
jasný ← den
("bright day")
```

```
jasný ← N
("bright NOUN")
```

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

```
V A A A A N J N V C
```

"It was a bright cold day in April and the clocks were striking thirteen"
Edge-Factored Parsers

• 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"
Edge-Factored Parsers

• Is this a good edge?

“...It was a bright cold day in April and the clocks were striking thirteen...”
Edge-Factored Parsers

- How about this competing edge?

“IT was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

- How about this competing edge?

```
not as good, lots of red ...
```

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

```
V A A A N J N V C
```

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

- How about this competing edge?

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

- 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

- 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

- 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 ...

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

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

• 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

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

- How about this competing edge?

\[ \text{jasný } \leftarrow \text{hodiny} \]

(“bright clocks”)

... undertrained ...

\[ \text{jasn } \leftarrow \text{hodi} \]

(“bright clock,” stems only)

\[ A_{\text{singular}} \leftarrow N_{\text{plural}} \]

V A A A N J N V C

byl jasn stud dubn den a hodi odbí třináctou

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

- How about this competing edge?

**jasný ↔ hodiny**
(“bright clocks”)  
... undertrained ...

**jasn ↔ hodi**
(“bright clock,”  
stems only)

**A_{singular} ↔ N_{plural}**

**“It was a bright cold day in April and the clocks were striking thirteen”**
Edge-Factored Parsers

- How about this competing edge?

\( \text{jasný} \leftarrow \text{hodiny} \)

\( \text{A} \leftarrow \text{N} \)

where \( \text{N} \) follows a conjunction

\( \text{jasn} \leftarrow \text{hodi} \)

(“bright clock,” stems only)

\( \text{A}_{\text{singular}} \leftarrow \text{N}_{\text{plural}} \)

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

V \ A \ A \ A \ N \ ] \ N \ V \ C

byl \ jasn \ stud \ dubn \ den \ a \ hodi \ odbí \ třín

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

• 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

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

“It was a bright cold day in April and the clocks were striking thirteen”
Edge-Factored Parsers

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

"It was a bright cold day in April and the clocks were striking thirteen"
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

```
Observed input sentence (shaded)
```

```
... find preferred tags ...
```
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

Possible tagging (i.e., assignment to remaining variables)

Observed input sentence (shaded)
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

Possible tagging (i.e., assignment to remaining variables)
Another possible tagging

Observed input sentence (shaded)
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

Possible tagging (i.e., assignment to remaining variables)
Another possible tagging

Observed input sentence (shaded)
Local factors in a graphical model

First, a familiar example

- Conditional Random Field (CRF) for POS tagging

"Binary" factor that measures compatibility of 2 adjacent tags
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

“Binary” factor that measures compatibility of 2 adjacent tags

Model reuses same parameters at this position

```
<table>
<thead>
<tr>
<th>v</th>
<th>n</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

![Diagram showing a graphical model with nodes labeled 'find', 'preferred', and 'tags' connected by edges. There is a box with values 0.2 and 0.2 associated with the 'tags' node.]

...
First, a familiar example
- Conditional Random Field (CRF) for POS tagging

"Unary" factor evaluates this tag

find
preferred
tags

| 0.2 | 0.2 | 0.2 | 0 |
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

"Unary" factor evaluates this tag
Its values depend on corresponding word
First, a familiar example

- Conditional Random Field (CRF) for POS tagging

"Unary" factor evaluates this tag
Its values depend on corresponding word

find

preferred
tags

can't be adj
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

"Unary" factor evaluates this tag
Its values depend on corresponding word
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

“Unary” factor evaluates this tag. Its values depend on corresponding word.

(find preferred tags)

(could be made to depend on entire observed sentence)
Local factors in a graphical model

First, a familiar example

- Conditional Random Field (CRF) for POS tagging

```
find  preferred  tags
```

```
\begin{array}{c|c|c}
\text{find} & 0.3 & \text{preferred} \\
\text{preferred} & 0.2 & \text{tags} \\
\end{array}
```

```
\begin{array}{c|c|c}
v & 0.3 & n \\
n & 0.02 & a \\
a & 0 & 0.1 \\
\end{array}
```

```
\begin{array}{c|c|c}
v & 0.3 & n \\
n & 0 & a \\
a & 0.2 & 0 \\
\end{array}
```
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

"Unary" factor evaluates this tag
Different unary factor at each position

```
find  preferred  tags

- 0.3  0.02
- 0.3  0.1
- 0.2  0.2
```
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

![Diagram with nodes and edges representing a CRF model for POS tagging. Each node has a label (v, n, a) and associated probabilities (0.3, 0.02, 0.2) for the tags 'find', 'preferred', and 'tags', respectively. The diagram illustrates the connectivity and the influence of local factors on the tagging process.]
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

\[ p(v, a, n) \] is proportional to the product of all factors' values on \( v, a, n \)

\[
p(v, a, n) \propto \text{product of all factors' values on } v, a, n
\]
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

\[ p(v a n) \] is proportional to the product of all factors' values on \( v a n \)

\[ p(v a n) \text{ is proportional to the product of all factors' values on } v a n \]
Local factors in a graphical model

- First, a familiar example
  - Conditional Random Field (CRF) for POS tagging

\[ p(v_{an}) \text{ is proportional to the product of all factors' values on } v_{an} \]

\[ v_{an} = \ldots 1 \times 3 \times 0.3 \times 0.1 \times 0.2 \ldots \]
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

... find preferred links ...
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

... find preferred links ...
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

... find preferred links ...
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

Possible parse...

... find preferred links ...
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

Possible parse...

... find preferred links ...

... and variable assignment
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

Another parse...

... find preferred links ...
Graphical Models for Parsing

• First, a labeling example
  ✤ CRF for POS tagging

• Now let’s do dependency parsing!
  ✤ $O(n^2)$ boolean variables for the possible links

Another parse...

... find preferred links ...

... and variable assignment
Graphical Models for Parsing

- First, a labeling example
  - CRF for POS tagging

- Now let’s do dependency parsing!
  - $O(n^2)$ boolean variables for the possible links

An illegal parse...

... find preferred links ...

... with a cycle!
Graphical Models for Parsing

• First, a labeling example
  ✤ CRF for POS tagging

• Now let’s do dependency parsing!
  ✤ $O(n^2)$ boolean variables for the possible links

An illegal parse...

... find preferred links ...
Local Factors for Parsing

• What factors determine parse probability?
Local Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
Local Factors for Parsing

- What factors determine parse probability?
  * Unary factors to score each link in isolation

... find preferred links ...
Local Factors for Parsing

• What factors determine parse probability?
  ✤ Unary factors to score each link in isolation

• But what if the best assignment isn’t a tree?

... find preferred links ...
Global Factors for Parsing

• What factors determine parse probability?
  ✤ Unary factors to score each link in isolation
Global Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to require links to form a legal tree

... find preferred links ...
Global Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to require links to form a legal tree
  - A hard constraint: potential is either 0 or 1

... find preferred links ...
Global Factors for Parsing

- What factors determine parse probability?
  - Unary factors to score each link in isolation
  - Global TREE factor to require links to form a legal tree
  - A hard constraint: potential is either 0 or 1

... find preferred links ...

... we're legal!

| f f f f f f | 0 |
| f f f f f t | 0 |
| f f f f t f | 0 |
| ... t f f f t | ... |
| ... f f f f t | ... |
| ... t t t t t t | ... |
| t t t t t t t t | 0 |
Global Factors for Parsing

- What factors determine parse
  - Unary factors to score each link in isolation
  - Global TREE factor to require links to form a legal tree
  - A hard constraint: potential is either 0 or 1

So far, this is equivalent to edge_factored parsing

... find preferred links ...
Global Factors for Parsing

- What factors determine parse
  - Unary factors to score each link in isolation
  - Global TREE factor to require links to form a legal tree
  - A hard constraint: potential is either 0 or 1

So far, this is equivalent to edge-factored parsing

... fir Note: traditional parsers don't loop through this table to consider exponentially many trees one at a time.
They use combinatorial algorithms; so should we!
Local Factors for Parsing

• What factors determine parse probability?
  ✤ Unary factors to score each link in isolation
  ✤ Global TREE factor to *require* links to form a legal tree
  • A *hard constraint*: potential is either 0 or 1
  ✤ Second order effects: factors on 2 variables
  • Grandparent–parent–child chains

... find preferred links ...
Local Factors for Parsing

• What factors determine parse probability?
  ✤ Unary factors to score each link in isolation
  ✤ Global TREE factor to require links to form a legal tree
  • A hard constraint: potential is either 0 or 1
  ✤ Second order effects: factors on 2 variables
  • Grandparent–parent–child chains

... find preferred links ...
Local Factors for Parsing

• What factors determine parse probability?
  ✤ Unary factors to score each link in isolation
  ✤ Global TREE factor to require links to form a legal tree
    • A hard constraint: potential is either 0 or 1
  ✤ Second order effects: factors on 2 variables
    • Grandparent–parent–child chains
    • No crossing links
    • Siblings
  ✤ Hidden morphological tags
  ✤ Word senses and subcategorization frames

... find preferred links ...
Great Ideas in ML: Message Passing

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

*Count the soldiers*

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

there's 1 of me

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

there's 1 of me

5 behind you
4 behind you
3 behind you
2 behind you
1 behind you

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

adapted from MacKay (2003) textbook
Count the soldiers

Belief:
Must be
2 + 1 + 3 = 6
of us

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

there’s 1 of me

Belief:
Must be 2 + 1 + 3 = 6 of us

1 before you

only see my incoming messages

4 behind you

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Count the soldiers

there's 1 of me

1 before you

only see my incoming messages

Belief: Must be 1 + 1 + 4 = 6 of us

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

*Each soldier receives reports from all branches of tree*

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

*Each soldier receives reports from all branches of tree*

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of the tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

*Each soldier receives reports from all branches of the tree*

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

*Each soldier receives reports from all branches of the tree*

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

*Each soldier receives reports from all branches of tree*

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Belief: Must be 14 of us

adapted from MacKay (2003) textbook
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree

Belief: Must be 14 of us

wouldn’t work correctly with a “loopy” (cyclic) graph

adapted from MacKay (2003) textbook
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward

![Diagram showing message passing in CRF](image-url)
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

- In the CRF, message passing = forward-backward
In the CRF, message passing = forward-backward
Great ideas in ML: Forward-Backward

\[ \alpha \quad \beta \]

\[ \begin{array}{c|c}
 v & 3 \\
 n & 1 \\
 a & 6 
\end{array} \quad \begin{array}{c|c}
 v & 2 \\
 n & 1 \\
 a & 7 
\end{array} \]

\[ \begin{array}{c|c|c}
 v & 0.3 \\
 n & 0 \\
 a & 0.1 
\end{array} \]

find \quad preferred \quad tags
Great ideas in ML: Forward-Backward

...  

find  preferred  tags

\[ \alpha \]

\[ \beta \]
Great ideas in ML: Forward-Backward

- Extend CRF to “skip chain” to capture non-local factor
Great ideas in ML: Forward-Backward

- Extend CRF to “skip chain” to capture non-local factor
  - More influences on belief 😊
Great ideas in ML: Forward-Backward

- Extend CRF to “skip chain” to capture non-local factor
  - More influences on belief 😊
  - Graph becomes loopy 🙁
Great ideas in ML: Forward-Backward

- Extend CRF to “skip chain” to capture non-local factor
  - More influences on belief 😊
  - Graph becomes loopy 😞

Red messages not independent? Pretend they are!

![Diagram showing forward-backward algorithm with nodes and connections.](image)
Great ideas in ML: Forward-Backward

- Extend CRF to “skip chain” to capture non-local factor
  - More influences on belief 😊
  - Graph becomes loopy 😞

Red messages not independent? Pretend they are!
“Loopy Belief Propagation”
Terminological Clarification

propagation
Terminological Clarification

belief → propagation
Terminological Clarification

loopy  belief  propagation
Terminological Clarification

\[\text{loopy} \quad \text{belief} \quad \text{propagation}\]
Terminological Clarification

(loopy belief) propagation
Terminological Clarification

\[
\text{loopy} \quad \text{belief} \quad \text{propagation}
\]
Terminological Clarification

(loopy belief propagation)

(loopy belief propagation)
Propagating Global Factors

• Loopy belief propagation is easy for local factors

• How do combinatorial factors (like TREE) compute the message to the link in question?
  ❖ “Does the TREE factor think the link is probably $t$ given the messages it receives from all the other links?”

... find preferred links ...
Propagating Global Factors

• Loopy belief propagation is easy for local factors

• How do combinatorial factors (like TREE) compute the message to the link in question?
  ▫ “Does the TREE factor think the link is probably \( t \) given the messages it receives from all the other links?”

... find preferred links ...
Propagating Global Factors

- Loopy belief propagation is easy for local factors
- How do combinatorial factors (like TREE) compute the message to the link in question?
  - “Does the TREE factor think the link is probably \( t \) given the messages it receives from all the other links?”

<table>
<thead>
<tr>
<th>TREE factor</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ffffffff</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fffftft</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fffttff</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ftttttt</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>tttttttt</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

... find preferred links ...
Propagating Global Factors

• How does the TREE factor compute the message to the link in question?

“Does the TREE factor think the link is probably $t$ given the messages it receives from all the other links?”

... find preferred links ...

<table>
<thead>
<tr>
<th>TREE factor</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ffffffff}$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{fffff}t$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{fffff}f$</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\text{ftffft}$</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\text{tttttt}$</td>
<td>0</td>
</tr>
</tbody>
</table>
Propagating Global Factors

• How does the TREE factor compute the message to the link in question?

❖ “Does the TREE factor think the link is probably \( t \) given the messages it receives from all the other links?”

Old-school parsing to the rescue!

This is the outside probability of the link in an edge-factored parser!

\[ \therefore \] TREE factor computes all outgoing messages at once (given all incoming messages)

Projective case: total \( O(n^3) \) time by inside-outside

Non-projective: total \( O(n^3) \) time by inverting Kirchhoff matrix
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!
**Kirchoff (Laplacian) Matrix**

\[
\begin{bmatrix}
0 & -s(1,0) & -s(2,0) & \cdots & -s(n,0) \\
0 & 0 & -s(2,1) & \cdots & -s(n,1) \\
0 & -s(1,2) & 0 & \cdots & -s(n,2) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & -s(1,n) & -s(2,n) & \cdots & 0 \\
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
0 & -s(1,0) & -s(2,0) & \cdots & -s(n,0) \\
0 & 0 & -s(2,1) & \cdots & -s(n,1) \\
0 & -s(1,2) & 0 & \cdots & -s(n,2) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & -s(1,n) & -s(2,n) & \cdots & 0 
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
0 & -s(1,0) & -s(2,0) & \cdots & -s(n,0) \\
0 & \sum_{j \neq 1} s(1,j) & -s(2,1) & \cdots & -s(n,1) \\
0 & -s(1,2) & \sum_{j \neq 2} s(2,j) & \cdots & -s(n,2) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & -s(1,n) & -s(2,n) & \cdots & \sum_{j \neq n} s(n,j)
\end{bmatrix}
\]

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

\[
\begin{bmatrix}
\sum_{j \neq 1} s(1, j) & -s(2,1) & \cdots & -s(n,1) \\
-s(1,2) & \sum_{j \neq 2} s(2, j) & \cdots & -s(n,2) \\
\vdots & \vdots & \ddots & \vdots \\
-s(1,n) & -s(2,n) & \cdots & \sum_{j \neq n} s(n, j)
\end{bmatrix}
\]

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

\[ \begin{bmatrix}
\sum_{j \neq 1} s(1, j) & -s(2,1) & \cdots & -s(n,1) \\
-s(1,2) & \sum_{j \neq 2} s(2, j) & \cdots & -s(n,2) \\
\vdots & \vdots & \ddots & \vdots \\
-s(1,n) & -s(2,n) & \cdots & \sum_{j \neq n} s(n, j)
\end{bmatrix} \]

- 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.*
Transition-Based Parsing

• Linear time
• Online
• Train a classifier to predict next action
• Deterministic or beam-search strategies
• But... generally less accurate
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Start state: \(([], [1, \ldots, n], \{\})\)

Final state: \((S, [], A)\)

Shift: \((S, i|B, A) \Rightarrow (S|i, B, A)\)

Reduce: \((S|i, B, A) \Rightarrow (S, B, A)\)

Right-Arc: \((S|i, j|B, A) \Rightarrow (S|i|j, B, A \cup \{i \rightarrow j\})\)

Left-Arc: \((S|i, j|B, A) \Rightarrow (S, j|B, A \cup \{i \leftarrow j\})\)
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack | Buffer | Arcs
--- | --- | ---
[ ]_s | [who, did, you, see]_B | {}
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack  Buffer  Arcs
[who]_S  [did, you, see]_B  {}
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack  Buffer  Arcs
[ ]s    [did, you, see]_B    { who \rightarrow OBJ did }

Left-arc
OBJ

who \rightarrow OBJ did \rightarrow SBJ you \rightarrow VG see
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack
[do]_S

Buffer
[you, see]_B

Arcs
{ who OBJ did }

Shift

OBJ

SBJ

who

did

you

see

VG
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack          Buffer          Arcs
[did, you]_S    [see]_B         { who  \overset{OBJ}{\longleftrightarrow}  did, \
                             did  \overset{SBJ}{\longrightarrow}  you }

Right-arc

SBJ

[diagram showing the transition-based parsing process]
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack  | Buffer  | Arcs
-------|---------|-------
[did]_S | [see]_B | { who \leftarrow^{OBJ} \text{did},
                  \text{did} \rightarrow^{SBJ} \text{you} }

Reduce

who \rightarrow^{OBJ} did \rightarrow^{SBJ} you \rightarrow^{VG} see
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack  Buffer  Arcs

\[ \text{[did, see]}_S \quad \text{[ ]}_B \quad \{ \text{who} \xrightarrow{\text{OBJ}} \text{did,} \]

\quad \text{did} \xrightarrow{\text{SBJ}} \text{you,} \quad \text{did} \xrightarrow{\text{VG}} \text{see} \}

Right-arc

VG

OBJ  SBJ  VG

who  did  you  see
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack

$[\text{did, you}]_S$

Buffer

$[\text{see}]_B$

Arcs

$\{ \text{who \ OBJ} \overset{\text{did, SBJ}}{\rightarrow} \text{did, you} \}$

Right-arc

$\text{SBJ}$

$\text{OBJ}$

$\text{VG}$

$\text{SBJ}$

$\text{who}$

$\text{did}$

$\text{you}$

$\text{see}$
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack
[did, you]_S

Buffer
[see]_B

Arcs
{ who \rightarrow OBJ did,
  did \leftarrow SBJ you }

Right-arc
SBJ

Choose action w/best classifier score
100k - 1M features
Transition-Based Parsing

Arc-eager shift-reduce parsing (Nivre, 2003)

Stack

[did, you]$_S$

Buffer

[S]

Very fast linear-time performance
WSJ 23 (2k sentences) in 3 s

did → you

Right-arc

SBJ

Choose action w/best classifier score
100k - 1M features

VG