Advances in Discriminative Dependency Parsing

Terry Koo
Dependency Grammar

Syntax represented by head-modifier dependencies

Applications include machine translation, semantic role labeling, etc.
**Discriminative Parsing**

- **Structured linear model** for parsing (McDonald, 2005):
  
  $$y^*(x; w) = \arg\max_{y \in \mathcal{Y}(x)} w \cdot \Phi(x, y)$$

- $x$ is a sentence, $\mathcal{Y}(x)$ is the set of possible trees
- Structures represented via feature mapping $\Phi(x, y)$
- Parameters $w$ provide a weight for each feature
- Direct maximization is generally **intractable**
Assume trees can be *factored into parts*.

Feature decomposition: \( \Phi(x, y) = \sum_{p \in y} \phi(x, p) \)
Factored Discriminative Parsing

Factored structured linear model:

\[ y^*(x; w) = \arg\max_{y \in \mathcal{Y}(x)} \sum_{p \in y} w \cdot \phi(x, p) \]

- Tractable for many factorizations
- Three main components:
  - **Feature mapping**: what features appear in \( \phi \)?
  - **Parameters**: how do we estimate \( w \)?
  - **Factorization**: which parts \( p \) make up \( y \)?
Lexicalized Representations

- Statistical parsers make heavy use of lexicalized features
- Alternate lexical representations?
Lexicalized Representations

- Statistical parsers make heavy use of lexicalized features
- Alternate lexical representations?
Non-Projective Parsing

- Non-projective parsing allows *crossing dependencies*
- Frequent in languages like Czech, Dutch, etc.
- Non-projective parsing = maximum spanning-tree
  
  (McDonald et al., 2005)
Non-Projective Parsing

- Many parameter estimation methods depend on summations over $\mathcal{Y}(x)$
  - Baum-Welch algorithm for Hidden Markov Models
  - Conditional Random Fields
- Efficient algorithms exist for many types of structure
- Algorithms for non-projective trees?
Higher-Order Factorizations

- First-order factorization: individual dependencies
- Second-order factorization: pairs of dependencies
  - *Sibling* and *Grandchild* interactions
- Factorizations with larger sub-structures?

The child attended school yesterday
Outline

- Introduction
- Three advances in discriminative dependency parsing:
  \[ \arg\max_{y \in \mathcal{Y}(x)} \sum_{p \in y} w \cdot \phi(x, p) \]
  
  - *Simple and effective lexical representations*
  - Parameter estimation for non-projective parsing
  - Efficient third-order dependency parsers
- Conclusion
Dependency Parsing Features

- A dependency is a pair \((h, m)\), e.g., \((2, 5)\)
- Features are 0/1 indicators for words, parts of speech
  - \(\phi_1(x, h, m) = \left[ \text{“ate” } \rightarrow \text{“with”} \right] \)
  - \(\phi_2(x, h, m) = \left[ \text{“VBD” } \rightarrow \text{“IN”} \right] \)
Alternate Lexical Representations

- Intermediate lexical representations derived from unlabeled data: *word clusters*
- Clusters are easily incorporated as features
- Improvements in English and Czech parsing
- Previous work:
  - Brown et al. (1992): clustering algorithm, applied to language modeling
  - Miller et al. (2004): named-entity tagging with word clusters from the Brown algorithm
Brown Algorithm

- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
- Prefixes of bit strings yield clusterings
- Prefix length determines granularity
Brown Algorithm

Words merged according to contextual similarity

Paths in the hierarchy represented as *bit strings*

Prefixes of bit strings yield clusterings

Prefix length determines granularity
Brown Algorithm

- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
- Prefixes of bit strings yield clusterings
- Prefix length determines granularity
Brown Algorithm

- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
- Prefixes of bit strings yield clusterings
- Prefix length determines granularity
Brown Algorithm

- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
- Prefixes of bit strings yield clusterings
- Prefix length determines granularity
Brown Algorithm

- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
- Prefixes of bit strings yield clusterings
- Prefix length determines granularity
Brown Algorithm

- Words merged according to contextual similarity
- Paths in the hierarchy represented as *bit strings*
- Prefixes of bit strings yield clusterings
- Prefix length determines granularity
Brown Algorithm

- Examples of clusters from our English experiments

- 0101010101110011 constructed
  0101010101110011 elucidated
  0101010101110011 inhaled
  0101010101110011 rewritten

- 1001111111100 precious-metal
  100111111111100 grain-futures
  100111111111100 crude-oil-futures

Past Participle Verbs

Financial Categories
Cluster-based Features

- Feature mappings can include arbitrary information
- Two types of features:
  - *Baseline* features include words and POS
  - *Cluster-based* feature sets add information from clusters
Cluster-based Features
Cluster-based Features
Cluster-based Features

4-bit 4-bit 4-bit

6-bit 6-bit 6-bit
Cluster-based Features

- 4-bit
- 4-bit
- 4-bit
- 4-bit
- POS
- 4-bit
- 4-bit
- 4-bit
- POS
- 4-bit
- 4-bit
- POS
- 6-bit
- 6-bit
- 6-bit
- 6-bit
- POS
- 6-bit
- 6-bit
- POS
- 6-bit
- 6-bit
- POS
Cluster-based Features
Feature Pruning

- Cluster-based feature sets were very large
- Eliminate features using word frequency
  - Only use the top-800 most frequent words
  - Cluster-based features were not affected
Experiments

- English parsing (Penn Treebank)
- Czech parsing (Prague Dependency Treebank)
- Brown clustering algorithm (Liang, 2005)
- Averaged perceptron training
  - Second-order projective parsers (Carreras, 2007)
  - First-order max-spanning-tree (McDonald, 2005)
- Compare between baseline and cluster-based features
Baseline Comparison

<table>
<thead>
<tr>
<th>Parsing Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald (2006)</td>
<td>91.5</td>
</tr>
<tr>
<td>Baseline Features</td>
<td>92.0</td>
</tr>
</tbody>
</table>

- **Attachment score** on English test set
- Percent of words attached to correct head
- Baseline parser is state of the art
**English Parsing Results**

<table>
<thead>
<tr>
<th>Test Set</th>
<th>Baseline</th>
<th>Cluster-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sec 00</td>
<td>91.8</td>
<td>92.8 (+1.0)</td>
</tr>
<tr>
<td>Sec 01</td>
<td>92.5</td>
<td>93.3 (+0.8)</td>
</tr>
<tr>
<td>Sec 23</td>
<td>92.0</td>
<td>93.2 (+1.2)</td>
</tr>
<tr>
<td>Sec 24</td>
<td>90.9</td>
<td>91.9 (+1.0)</td>
</tr>
</tbody>
</table>

- Attachment score on all English test sets (Sections 00, 01, 23, and 24 of the Penn Treebank)
- Cluster-based features outperform baseline
## Effect of Training Corpus Size

<table>
<thead>
<tr>
<th>#Sentences</th>
<th>Baseline</th>
<th>Cluster-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>82.0</td>
<td>85.3 (+3.3)</td>
</tr>
<tr>
<td>2000</td>
<td>85.0</td>
<td>87.5 (+2.5)</td>
</tr>
<tr>
<td>4000</td>
<td>87.9</td>
<td>89.7 (+1.8)</td>
</tr>
<tr>
<td>8000</td>
<td>89.7</td>
<td>91.4 (+1.7)</td>
</tr>
<tr>
<td>16000</td>
<td>91.1</td>
<td>92.2 (+1.1)</td>
</tr>
<tr>
<td>32000</td>
<td>92.1</td>
<td>93.2 (+1.1)</td>
</tr>
<tr>
<td>39832</td>
<td>92.4</td>
<td>93.3 (+0.9)</td>
</tr>
</tbody>
</table>

- Attachment score on English development set
- Part-of-speech tagger trained on reduced dataset
# Sentences & Baseline | Cluster-Based
---|---
1000 | 82.0 | 85.3 (+3.3)
2000 | 85.0 | 87.5 (+2.5)
4000 | 87.9 | 89.7 (+1.8)
8000 | 89.7 | 91.4 (+1.7)
16000 | 91.1 | 92.2 (+1.1)
32000 | 92.1 | 93.2 (+1.1)
39832 | 92.4 | 93.3 (+0.9)

- Attachment score on English development set
- Part-of-speech tagger trained on reduced dataset
### Czech Parsing Results

<table>
<thead>
<tr>
<th>Parsing Model</th>
<th>Baseline</th>
<th>Cluster-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>First-order MST</td>
<td>84.5</td>
<td>86.1 (+1.6)</td>
</tr>
<tr>
<td>McDonald (2006) second-order</td>
<td>85.2</td>
<td></td>
</tr>
<tr>
<td>Second-order</td>
<td>86.1</td>
<td>87.1 (+1.0)</td>
</tr>
</tbody>
</table>

- Attachment score on Czech test set
- Results are similar to English
## Removal of Direct Lexicalization

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Baseline</th>
<th>Cluster-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>90.6 (-1.8)</td>
<td>93.1 (-0.2)</td>
</tr>
<tr>
<td>800</td>
<td>91.9 (-0.5)</td>
<td>93.3</td>
</tr>
<tr>
<td>All words</td>
<td>92.4</td>
<td></td>
</tr>
</tbody>
</table>

- Attachment score on English development set
- Cluster-based features are far less sensitive
Clusters vs Part-of-Speech Tags

<table>
<thead>
<tr>
<th></th>
<th>Ignore POS Tags</th>
<th>Use POS Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ignore Clusters</strong></td>
<td>86.7</td>
<td>92.4</td>
</tr>
<tr>
<td><strong>Use Clusters</strong></td>
<td>91.8</td>
<td>93.3</td>
</tr>
</tbody>
</table>

- Attachment score on English development set
- Clusters *alone* are almost as good as baseline
Summary

- Lexical statistics are important but sparse
- Word clusters as an alternate lexical representation
- Clusters incorporated as features for a discriminative parser
- Performance gains over a state-of-the-art baseline
Outline

- Introduction
- Three advances in discriminative dependency parsing:
  \[\arg\max_{y\in\mathcal{Y}(x)} \sum_{p\in y} w \cdot \phi(x, p)\]
  - Simple and effective lexical representations
  - *Parameter estimation for non-projective parsing*
  - Efficient third-order dependency parsers
- Conclusion
Non-Projective Inference

- Fundamental inference algorithms that sum over possible structures:

<table>
<thead>
<tr>
<th>Structured Model</th>
<th>Inference Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hidden Markov Model</td>
<td>Forward-Backward</td>
</tr>
<tr>
<td>Graphical Model</td>
<td>Belief Propagation</td>
</tr>
<tr>
<td>Context-Free Grammar</td>
<td>Inside-Outside</td>
</tr>
<tr>
<td>Projective Dependencies</td>
<td>Inside-Outside</td>
</tr>
<tr>
<td>Non-Projective Dependencies</td>
<td>???</td>
</tr>
</tbody>
</table>

- New inference algorithms for non-projective parsing
Log-Linear Dependency Parsers

- Distribution over trees in a first-order factorization

\[ P(y \mid x; w) \propto \prod_{(h,m) \in y} e^{w \cdot \phi(x,h,m)} \]

- Parsing is a search for the most probable tree

- A popular method for modeling structured data

- Also known as a Conditional Random Field (CRF)
Log-Linear Parameter Estimation

- Learn $\mathbf{w}$ from labeled data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

- Maximize (regularized) conditional log-likelihood:
  \[
  f_{LL}(\mathbf{w}) = -\frac{C}{2} \| \mathbf{w} \|^2 + \sum_{i=1}^{n} \log P(y_i | x_i; \mathbf{w})
  \]

- Gradient-based optimization
  - $f_{LL}(\mathbf{w})$ and $\nabla f_{LL}(\mathbf{w})$
  - e.g., L-BFGS
Log-Linear Inference Problems

- $f_{LL}(w)$ requires the partition function:
  \[ Z(x; w) = \sum_{y} \prod_{(h,m) \in y} e^{w \cdot \phi(x,h,m)} \]

- $\nabla f_{LL}(w)$ requires the marginal probabilities:
  \[ P(h, m \mid x; w) = \sum_{y : (h,m) \in y} P(y \mid x; w) \]
The Matrix-Tree Theorem

- Originally developed by Kirchhoff (1847)
- Count the number of undirected spanning trees
- Determinant of a specially-constructed matrix
- Extended by Tutte (1984)
- Summations over weighted, rooted, directed spanning trees
The Matrix-Tree Theorem

Given:

- Directed graph $G$
- Edge weights $\theta_{i,j}$
- Root node $r$

Construct a matrix $L^{(r)}$ such that:

$$\left| L^{(r)} \right| = \sum_{\text{Trees}} \prod_{\text{Edges in Tree}} e^{\theta_{h,m}}$$
The Matrix-Tree Theorem

- Given:
  - Directed graph $G$
  - Edge weights $\theta_{i,j}$
  - Root node $r$

- Construct a matrix $L^{(r)}$ such that:

$$\left| L^{(r)} \right| =$$
Given:
- Directed graph $G$
- Edge weights $\theta_{i,j}$
- Root node $r$

Construct a matrix $L^{(r)}$ such that:

$$\left| L^{(r)} \right| = e^{1.0+2.5} + e^{0.5+2.0}$$
The Partition Function

- A naive method: one invocation per root

- Inefficient: requires \( n \) determinants
The Partition Function

- A simple method for summing over all roots:

- The modifier of * is the root
The Partition Function

A simple method for summing over all roots: \( |L(*)| \)
The Partition Function

- A simple method for summing over all roots: \( |L^{(*)}| \)

\[
L^{(*)} = \begin{vmatrix}
L \quad (\ast)
\end{vmatrix}
\]

- Determinant of \( n \times n \) matrix: \( O(n^3) \)
Multi-Root Dependency Trees

A simple method for summing over all roots: \(|L(\ast)|\)

Structures with *multiple* roots are counted
Single-Root Partition Function

A new matrix for summing over single-root trees:

\[ \hat{L} = \begin{cases} 
\hat{L}_{1,j} = e^{\theta_{*},j} & \\
\hat{L}_{i,j} = -e^{\theta_{i},j} & \\
\hat{L}_{j,j} = \sum_{i \neq *} e^{\theta_{i},j} 
\end{cases} \]

Determinant of \( n \times n \) matrix: \( O(n^3) \)
Marginal Probabilities

- Single-root and multi-root partition functions:
  \[ Z(\theta) = |\hat{L}| \quad Z(\theta) = |L^{(*)}| \]

- Marginals are derivatives of log partition function:
  \[ P(h \rightarrow m; \theta) = \frac{\partial \log Z(\theta)}{\partial \theta_{h,m}} \]

- Derivative of log-determinant:
  \[ \frac{\partial \log |X|}{\partial X} = (X^{-1})^T \]

- Inverse of \( n \times n \) matrix: \( O(n^3) \)
Application to Parsing

- Training a log-linear parser:
  - Define edge scores $\theta_{h,m} = w \cdot \phi(x, h, m)$
  - Construct appropriate matrix $\hat{L}$ or $L^{(*)}$
  - $Z(x; w)$ via matrix determinant
  - $P(h, m | x; w)$ via matrix inverse
- Max-margin training for dependency parsers
- Exponentiated Gradient (Collins et al., 2008)
Multilingual Parsing Experiments

- Six languages from CoNLL-X shared task
- Three training algorithms:
  - Averaged perceptron
  - Log-linear models
  - Max-margin models
- Projective and non-projective parsing
Dutch Parsing Experiments

- Attachment score on Dutch test set
- 4.93% of dependencies are crossing
- Non-projective training is beneficial for languages with non-projectivity

<table>
<thead>
<tr>
<th>Training Algorithm</th>
<th>Projective Training</th>
<th>Non-Projective Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron</td>
<td>77.2</td>
<td>78.8 (+1.6)</td>
</tr>
<tr>
<td>Log-Linear</td>
<td>76.2</td>
<td>79.6 (+3.4)</td>
</tr>
<tr>
<td>Max-Margin</td>
<td>76.5</td>
<td>79.7 (+3.2)</td>
</tr>
</tbody>
</table>
## Aggregate Multilingual Results

<table>
<thead>
<tr>
<th>Training Algorithm</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perceptron</td>
<td>79.1</td>
</tr>
<tr>
<td>Log-Linear</td>
<td>79.7 (+0.6)</td>
</tr>
<tr>
<td>Max-Margin</td>
<td>79.8 (+0.7)</td>
</tr>
</tbody>
</table>

- Cumulative attachment score over 6 languages:
  - Arabic, Dutch, Japanese, Slovene, Spanish, Turkish
- Improvements are statistically significant
Summary

- New algorithms for weighted summations over non-projective dependency trees
- Covering both single-root and multi-root trees
- Efficient $O(n^3)$ algorithms
- An application: log-linear and max-margin parsers
Outline

- Introduction
- Three advances in discriminative dependency parsing:
  \[ \arg\max_{y \in \mathcal{Y}(x)} \sum_{p \in y} w \cdot \phi(x, p) \]
  - Simple and effective lexical representations
  - Parameter estimation for non-projective parsing
  - \textit{Efficient third-order dependency parsers}
- Conclusion
Higher-Order Parsers

<table>
<thead>
<tr>
<th>Parsing Approach</th>
<th>First-Order</th>
<th>Second-Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald’s Models</td>
<td>90.9</td>
<td>91.5 (+0.6)</td>
</tr>
<tr>
<td>Baseline Features</td>
<td>90.8</td>
<td>92.0 (+1.2)</td>
</tr>
<tr>
<td>Cluster-Based Features</td>
<td>92.2</td>
<td>93.2 (+1.0)</td>
</tr>
</tbody>
</table>

- Attachment scores on English test set
- Can we get more by going \textit{beyond} second-order?
- How much will it cost to get there?
- Carreras (2007) second-order is already $O(n^4)$
Third-Order Factorizations

- Two axes: Vertical context and Horizontal context

First-Order Parsers

h m dependency
Third-Order Factorizations

Two axes: Vertical context and Horizontal context

McDonald (2006) Second-Order

dependency

sibling
Third-Order Factorizations

Two axes: Vertical context and Horizontal context

Carreras (2007) Second-Order
Third-Order Factorizations

- Two axes: Vertical context and Horizontal context

Third-Order Model 1

- grandchild: g, h, m
- dependency: h, m
- grand-sibling: g, h, s, m
- sibling: h, s, m
Third-Order Factorizations

- Two axes: Vertical context and Horizontal context

- Grandchild
- Grand-sibling
- Dependency
- Sibling
- Tri-sibling

Third-Order Model 2
First-Order Parsing Algorithm

- Eisner (2000) algorithm: $O(n^3)$

**Complete Span**
A “half-constituent”

**Incomplete Span**
A dependency
First-Order Parsing Algorithm

- Eisner (2000) algorithm: $O(n^3)$

- Derivation of *complete* and *incomplete* spans:
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$
First-Order Parsing Example

- Eisner (2000) algorithm: $O(n^3)$
Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$
- Introduce a third type of span:

  **Sibling Span**
  A pair of adjacent modifiers

\[
\begin{array}{c}
S \\
\hline
m
\end{array}
\]
Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$

Scores sibling interactions
Second-Order Parsing Algorithm

- McDonald (2006) and Eisner (1996): $O(n^3)$

Scores sibling interactions
Second-Order Parsing Algorithm

McDonald (2006) and Eisner (1996): $O(n^3)$

\[
\begin{align*}
h & \quad e \\
\quad & = \\
\begin{array}{c}
\begin{array}{c}
\quad \\
\quad
\end{array}
\end{array} & \quad \\
\begin{array}{c}
\begin{array}{c}
\quad \\
\quad
\end{array}
\end{array} & \quad + \\
\begin{array}{c}
\begin{array}{c}
\quad \\
\quad
\end{array}
\end{array} & \quad \\
\begin{array}{c}
\begin{array}{c}
\quad \\
\quad
\end{array}
\end{array}
\end{align*}
\]
Model 0

- Model 0, all grandparents: $O(n^4)$

**Complete G-Span**
A “half-constituent”
with its grandparent

**Incomplete G-Span**
A dependency
with its grandparent

- Superficially similar to parent annotation in CFGs
Model 0: Derivations

- Model 0, all grandparents: $O(n^4)$

- Grandparent indices propagated to smaller g-spans

- 4 active indices, runtime $O(n^4)$
Model 1

- Model 1, grand-siblings: \( O(n^4) \)
- Introduce a third type of span:

**Sibling G-Span**
A pair of adjacent modifiers with their shared head
Model 1: Grand-Sibling Scores

- **Model 1, grand-siblings**: $O(n^4)$

- Scores grand-sibling interactions
Model 1: Grand-Sibling Scores

- Model 1, grand-siblings: $O(n^4)$

- Scores grand-sibling interactions
Model 1: Derivations

Model 1, grand-siblings: $O(n^4)$
Model 2

- Model 2, grand-siblings and tri-siblings: $O(n^4)$
- Introduce a fourth type of span:

**Incomplete S-Span**

A dependency with its next-inner modifier

[Diagram of an incomplete S-Span with labels $h$, $s$, and $m$.]
Model 2

- Model 2, grand-siblings and tri-siblings: $O(n^4)$
- Capable of recovering all tri-siblings:
- And some grand-siblings:
Summary of Parsing Algorithms

- Model 0 parses an all-grandchildren factorization
- Model 1 parses an all-grand-siblings factorization
- Model 2 parses all-tri-siblings and some grand-siblings
- All parsers require $O(n^4)$ time and $O(n^3)$ space
- Identical to Carreras (2007) second-order
- Models 1 and 2 have optimal runtime
- Total number of third-order parts: $O(n^4)$
### English and Czech Parsing

<table>
<thead>
<tr>
<th>Parser</th>
<th>English</th>
<th>Czech</th>
</tr>
</thead>
<tbody>
<tr>
<td>McDonald (2006)</td>
<td>91.5</td>
<td>85.2</td>
</tr>
<tr>
<td>Second-order, Baseline</td>
<td>92.0</td>
<td>86.1</td>
</tr>
<tr>
<td>Model 1</td>
<td>93.0</td>
<td>87.4</td>
</tr>
<tr>
<td>Model 2</td>
<td>92.9</td>
<td>87.4</td>
</tr>
<tr>
<td>Second-order, Clusters</td>
<td>93.2</td>
<td>87.1</td>
</tr>
</tbody>
</table>

- Attachment score on the English and Czech test sets
- Third-order comparable to *semi-supervised*
Summary

- Third-order factorizations can be parsed in $O(n^4)$
- Third-parsers work well in practice
- Possible extensions:
  - Recovering word senses or dependency labels
  - Increasing context to fourth-order or more
  - Using cluster-based features
Outline

- Introduction
- Three advances in discriminative dependency parsing:
  \[
  \arg\max_{y \in \mathcal{Y}(x)} \sum_{p \in y} w \cdot \phi(x, p)
  \]
  - Simple and effective lexical representations
  - Parameter estimation for non-projective parsing
  - Efficient third-order dependency parsers
- Conclusion
Conclusions

- Dependency parsing is a simple and effective framework for syntactic analysis
- Structured linear models provide three opportunities for improvements
  - Feature representations
  - Parameter estimation
  - Factorization