Dependency Parsing

Emily Pitler – Google

Thanks to:
Slav Petrov, Dan Klein, Ryan McDonald, Alexander Rush, Joakim Nivre,
David Weiss
Statistical Natural Language Processing, NYU, Fall 2015

Dependency Parsing

Input: Sequence of words
Output: Directed tree

A “real” Sentence

Influential members of the House Ways and Means Committee introduced legislation that would restrict how the new savings-and-loan bailout agency can raise capital, creating another potential obstacle to the government's sale of sick thrifts.
CoNLL Format

1 Cathy   Cathy   N N   eigen|ev|nout  2 eu
2 was     sle     V V   trans|en|vl|ls|203|v   0 root
3 hen     hen     Pron Pron   pron|en|3|v|dataface  2 obj1
4 wild    wild    Adj Adj   attc|en|b|o|onerv  5 mod
5 zwaaien zwaaien N N   assrt|en|b|o|newt  2 vc
6 .       .       Punc Punc  punct  5 punct

* Cathy zag hen wild zwaaien .

0 1 2 3 4 5 6

http://ilk.uvt.nl/conll/

Translation

English parses

Chinese ordering

cats [with whiskers and tails] → [with whiskers and tails] cats

cats [with whiskers] and tails → [with whiskers] cats and tails

Example from Huang, 1983

Information Extraction

Road rage leads police to murder victim's boyfriend

Difficulties: Attachments

- I cleaned the dishes from dinner
- I cleaned the dishes with detergent
- I cleaned the dishes in my pajamas
- I cleaned the dishes in the sink
**Difficulties: (Non-)Projectivity**

- Crossing Arcs needed to account for non-projective constructions
- Fairly rare in English but can be common in other languages (e.g. Czech):

```
root John saw a dog yesterday which was a Yorkshire Terrier
```

```
root O to nově většinou nemá ani zřejmě a taky na to většinou nemá peníze
```

**He is mostly not even interested in the new things and in most cases, he has no money for it either.**

**Formal Conditions**

- Directed tree (always)
  - Acyclic
  - Non-roots have exactly 1 incoming arc

- Projectivity (sometimes)
  - Each subtree is a contiguous interval
  - No arcs cross

**Styles of Dependency Parsing**

- **Transition-Based (tr)**
  - Fast, greedy, linear time inference algorithms
  - Trained for greedy search
  - Beam search

- **Graph-Based (gr)**
  - Slower, exhaustive, dynamic programming inference algorithms
  - Higher-order factorizations

**Time**

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>[Nivre et al. '03-'11]</th>
<th>[McDonald et al. '05-'06]</th>
</tr>
</thead>
<tbody>
<tr>
<td>greedy tr</td>
<td>( O(n) )</td>
<td></td>
</tr>
<tr>
<td>1st-order gr</td>
<td>( O(n^3) )</td>
<td></td>
</tr>
<tr>
<td>k-best tr</td>
<td>( O(k \cdot n) )</td>
<td></td>
</tr>
<tr>
<td>2nd-order gr</td>
<td>( O(n^4) )</td>
<td></td>
</tr>
<tr>
<td>3rd-order gr</td>
<td>( O(n^4) )</td>
<td></td>
</tr>
<tr>
<td>nn k-best tr</td>
<td>( O(n^3) )</td>
<td></td>
</tr>
</tbody>
</table>

**Accuracy**

<table>
<thead>
<tr>
<th>[Nivre et al. '03-'11]</th>
<th>[McDonald et al. '05-'06]</th>
</tr>
</thead>
<tbody>
<tr>
<td>greedy tr</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>1st-order gr</td>
<td>( O(n^3) )</td>
</tr>
<tr>
<td>k-best tr</td>
<td>( O(k \cdot n) )</td>
</tr>
<tr>
<td>2nd-order gr</td>
<td>( O(n^4) )</td>
</tr>
<tr>
<td>3rd-order gr</td>
<td>( O(n^4) )</td>
</tr>
<tr>
<td>nn tr</td>
<td>( O(n^3) )</td>
</tr>
</tbody>
</table>

\[\text{[Nivre et al. '03-'11]} \quad \text{Time} \quad \text{[McDonald et al. '05-'06]}\]
Graph-based Parsing

- Assumes that scores factor over the tree
- Arc-factored models
  - Score(tree) = Σ edges

\[ I \text{ washed dishes with detergent} = I \text{ washed} + \]
\[ \text{washed dishes} + \]
\[ \text{washed with} + \]
\[ \text{with detergent} \]

Dependency Representation

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

* Parsing algorithm
  - Set of possible trees
  - Factorization: local parts of the tree

* Searching
  - Scoring

Heads

Modifiers
Formal Conditions

• Directed tree (always)
  • Acyclic
  • Non-roots have exactly 1 incoming arc

• Projectivity (sometimes)
  • Each subtree is a contiguous interval
  • No arcs cross

Arc-factored Projective Parsing

• All projective graphs can be written as the combination of two smaller adjacent graphs

Arc-factored Projective Parsing

► Chart item filled in a bottom-up manner
  • First do all strings of length 1, then 2, etc., just like CKY

► Weight of new item: \( \max_{j,k} w(A) \times w(B) \times w_{hk} \)

► Algorithm runs in \( O(|L|n^5) \)

► Use back-pointers to extract best parse (like CKY)

Eisner Algorithm

► \( O(|L|n^5) \) is not that good

► [Eisner 1996] showed how this can be reduced to \( O(|L|n^3) \)
  • Key: split items so that sub-roots are always on periphery
Eisner First-Order Parsing

In practice also left arc version

Eisner First-Order Parsing

As McGwire neared, fans went wild

Eisner First-Order Parsing

As McGwire neared, fans went wild

Eisner First-Order Parsing

As McGwire neared, fans went wild
As McGwire neared, fans went wild.
**Eisner First-Order Parsing**

As McGwire neared, fans went wild.

---

**Eisner Algorithm Pseudo Code**

Initialization: \( C[s][a][d][c] = 0 \) \( \forall s, d, c \)

for \( k : 1..n \)
   for \( s : 1..n \)
      \( t = s + k \)
      if \( t > n \) then break

% First: create incomplete items
\( C[s][t][\rightarrow][0] = \max_{s\leq r<1} (C[s][r][\rightarrow][1] + C[r+1][t][\rightarrow][0] + s(t, s)) \)
\( C[s][t][\rightarrow][0] = \max_{s\leq r<1} (C[s][r][\rightarrow][1] + C[r+1][t][\rightarrow][0] + s(s, t)) \)

% Second: create complete items
\( C[s][t][\rightarrow][\rightarrow][1] = \max_{s\leq r<1} (C[s][r][\rightarrow][1] + C[r][t][\rightarrow][0]) \)
\( C[s][t][\rightarrow][\rightarrow][1] = \max_{s\leq r<1} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1]) \)

end for
end for

---

**Maximum Spanning Trees (MSTs)**

- A directed spanning tree of a (multi-)digraph \( G = (V, A) \), is a subgraph \( G' = (V', A') \) such that:
  - \( V' = V \)
  - \( A' \subseteq A \), and \( |A'| = |V'|-1 \)
  - \( G' \) is a tree (acyclic)

- A spanning tree of the following (multi-)digraphs

Can use MST algorithms for nonprojective parsing!
**Chu-Liu-Edmonds**

- $x = \text{root}$  
  John saw Mary

```
root 10
9 20 saw 30
John 30 0 Mary
11 3
```

**Chu-Liu-Edmonds**

- Find highest scoring incoming arc for each vertex

```
root
20 saw 30
John 30 Mary
```

- If this is a tree, then we have found MST!!

**Find Cycle and Contract**

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle

```
root 40
9 saw 30
John w_j s Mary
31
```

**Recalculate Edge Weights**

- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$ is 40 (**)
  - $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$ is 29
Theorem

The weight of the MST of this contracted graph is equal to the weight of the MST for the original graph.

\[ \text{root} \quad 40 \]
\[ \text{saw} \quad 9 \]
\[ \text{John} \quad w_{js} \]
\[ \text{Mary} \]

Therefore, recursively call algorithm on new graph.

Final MST

- This is a tree and the MST for the contracted graph!!

\[ \text{root} \quad 40 \]
\[ \text{saw} \quad 30 \]
\[ \text{John} \]
\[ \text{Mary} \]

- Go back up recursive call and reconstruct final graph.

Chu-Liu-Edmonds PseudoCode

Chu-Liu-Edmonds\((G_s, w)\)
1. Let \( M = \{(i^*, j) : j \in V_s, i^* = \arg \max_i w_{ij} \} \)
2. Let \( G_M = (V_s, M) \)
3. If \( G_M \) has no cycles, then it is an MST: return \( G_M \)
4. Otherwise, find a cycle \( C \) in \( G_M \)
5. Let \( G_C = \text{Chu-Liu-Edmonds}(G_C, w) \)
6. Let \( G = \text{Chu-Liu-Edmonds}(G, w) \)
7. Find vertex \( i \in C \) such that \((i', i) \in G\) and \( ma(i', i) = i \)
8. Find arc \((i'', i) \in C\)
9. Find all arc \((c, i'') \in G\)
10. \( G = G \cup \{(ma(c, i''), i') : (c, i'') \in G \cup C \cup \{(i', i)\} \}
11. Remove all vertices and arcs in \( G \) containing \( c \)
12. return \( G \)

- Reminder: \( w_{ij} = \arg \max_k w_{ij}^k \)

Chu-Liu-Edmonds PseudoCode

contract\((G = (V, A), C, w)\)
1. Let \( G_C \) be the subgraph of \( G \) excluding nodes in \( C \)
2. Add a node \( c \) to \( G_C \) representing cycle \( C \)
3. For \( i \in V - C \): \( \exists \in C \) \( (i', i) \in A \)
   Add arc \((c, i) \) to \( G_C \) with
   \( ma(c, i) = \arg \max_{i' \in C} \text{score}(i', i) \)
   \( i' = ma(c, i) \)
   \( \text{score}(c, i) = \text{score}(i', i) \)
4. For \( i \in V - C \): \( \exists \in C \) \( (i', i) \in A \)
   Add edge \((i, c) \) to \( G_C \) with
   \( ma(i, c) = \arg \max_{i' \in C} [\text{score}(i, i') - \text{score}(a(i'), i')] \)
   \( i' = ma(i, c) \)
   \( \text{score}(i, c) = [\text{score}(i, i') - \text{score}(a(i'), i')] + \text{score}(C) \)
   where \( a(v) \) is the predecessor of \( v \) in \( C \)
   and \( \text{score}(C) = \sum_{v \in C} \text{score}(a(v), v) \)
5. return \( G_C, c, ma \)
CLE: No Sequence Constraints

- I washed dishes with detergent
- dishes detergent I washed with

Search Spaces for Parsing

Directed Spanning Trees
NP-hard with models beyond arc-factored
(McDonald and Pereira, 2006; McDonald and Satta, 2007)
Mildly Non-Projective

Projective = 1 Interval / Subtree

- Chart item filled in a bottom-up manner
  - First do all strings of length 1, then 2, etc. just like CKY
- Weight of new item: $\max_{i,j,k} w(A) \times w(B) \times w_{h'j}$
- Algorithm runs in $O(|L|^3)$
- Use back-pointers to extract best parse (like CKY)
2 Intervals per Subtree

- Chart item filled in a bottom-up manner

- Algorithm runs in time $O(n^8)$
  - with a little work, $O(n^7)$ (Gómez-Rodríguez et al, 2011)

1-Endpoint-Crossing Trees

1-Endpoint Crossing Trees

∀ crossed edges $e$

∃ common point $p$

∀ edges that cross $e$ are incident to $p$

* Who do you think arrived in NY?

Not Arbitrary Sequences

<table>
<thead>
<tr>
<th>Language</th>
<th>1 Interval per Subtree</th>
<th>2 Intervals per Subtree</th>
<th>1-Endpoint-Crossing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>96.6</td>
<td>96.8</td>
<td>97.3</td>
</tr>
<tr>
<td>Czech</td>
<td>99.5</td>
<td>98.9</td>
<td>99.4</td>
</tr>
<tr>
<td>Portuguese</td>
<td>95.4</td>
<td>99.3</td>
<td>99.7</td>
</tr>
<tr>
<td>Danish</td>
<td>99.7</td>
<td>99.1</td>
<td>99.2</td>
</tr>
<tr>
<td>Swedish</td>
<td>99.2</td>
<td>98.7</td>
<td>98.7</td>
</tr>
</tbody>
</table>

Can Have Many Intervals

- Below is 1-Endpoint Crossing

Subtree rooted at $b$
Chart: 1 Interval + 1 Vertex

Only Periphery Can Have Edges Outside

Is This Purple Arc Allowed?

No
Is This Purple Arc Allowed?

Yes

1-Endpoint Crossing Parsing

- $n^2$ items built with 2 split points
- Algorithm runs in time $O(n^3)$

1-Endpoint Crossing Parsing

- $n^3$ items built with 1 split point
- Algorithm runs in time $O(n^4)$

Parsing Improvements

Unlabeled Attachment Accuracy (UAS)

- Projective GrandSib
- 1-Endpoint-Crossing Crossing-Sensitive GrandSib

<table>
<thead>
<tr>
<th></th>
<th>Dutch</th>
<th>Czech</th>
<th>Portuguese</th>
<th>Danish</th>
<th>Swedish</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAS</td>
<td>80</td>
<td>84</td>
<td>89</td>
<td>92</td>
<td>83</td>
</tr>
</tbody>
</table>

CoNLL shared task data, normalized with HamleDT, Stanford conjunctions (Zeman et al., 2012)
Graph-based Parsing

Arc Weights

\[ w^{k}_{ij} = e^{f(i,j,k)} \]

- Arc weights are a linear combination of features of the arc, \( f \), and a corresponding weight vector \( w \)
- Raised to an exponent (simplifies some math ...)
- What arc features?
- [McDonald et al. 2005] discuss a number of binary features

Arc Feature Ideas for \( f(i,j,k) \)

- Identities of the words \( w_i \) and \( w_j \) and the label \( l_k \)
- Part-of-speech tags of the words \( w_i \) and \( w_j \) and the label \( l_k \)
- Part-of-speech of words surrounding and between \( w_i \) and \( w_j \)
- Number of words between \( w_i \) and \( w_j \), and their orientation
- Combinations of the above

First-Order Feature Computation

- Identify PP
- John saw Mary McGuire yesterday with his telescope
  - \( N \ V \ N \ N \ R \ P \ PR \ N \)
(Structured) Perceptron

Training data: \( T = \{(x_t, G_t)\}_{t=1}^{|T|} \)
1. \( w^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( G' = \arg \max_{G'} w^{(i)} \cdot f(G') \)
5. if \( G' \neq G_t \)
6. \( w^{(i+1)} = w^{(i)} + f(G_t) - f(G') \)
7. \( i = i + 1 \)
8. return \( w^i \)

Styles of Dependency Parsing

- Transition-Based (tr)
  - Fast, greedy, linear time inference algorithms
  - Trained for greedy search
  - Beam search

- Graph-Based (gr)
  - Slower, exhaustive, dynamic programming inference algorithms
  - Higher-order factorizations

**Time**
- greedy tr: \( O(n) \)
- 1st-order gr: \( O(n^3) \)
- 2nd-order gr: \( O(n^4) \)
- 3rd-order gr: \( O(n^k) \)

**Accuracy**
- nn tr: \( O(n) \)
- nn k-best tr: \( O(n^k \cdot n) \)

\[ \text{[Nivre et al. '03-'11]} \quad \text{[McDonald et al. '05-'06]} \]

Transition Based Dependency Parsing

- Process sentence left to right
  - Different transition strategies available
  - Delay decisions by pushing on stack

- Arc-Standard Transition Strategy [Nivre '03]

  Initial configuration: ([], [0,…,n], [])
  Terminal configuration: ([0], [], A)

  shift: \( (\sigma,[i|\beta],A) \rightarrow ([\sigma][i],\beta,A) \)
  left-arc (label): \( ([\sigma][i][j], B,A) \rightarrow ([\sigma][j], B, A \cup \{j,i\}) \)
  right-arc (label): \( ([\sigma][i][j], B,A) \rightarrow ([\sigma][i], B, A \cup \{i,j\}) \)

Arc-Standard Example

\[ \begin{array}{c} \leftarrow \text{Buffer} \\ I \quad \text{booked} \quad a \quad \text{flight} \quad \text{to} \quad \text{Lisbon} \end{array} \]

\[ \uparrow \text{Stack} \]

I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon

I booked a flight to Lisbon

I booked a flight to Lisbon

I booked a flight to Lisbon
I booked a flight to Lisbon.

Stack top word = "flight"  
Stack top POS tag = "NOUN"  
Buffer front word = "to"  
Child of stack top word = "a"  

SHIFT  
RIGHT-ARC?  
LEFT-ARC?
SVM / Structured Perceptron Hyperparameters

- Regularization
- Loss function
- Hand-crafted features

Features ZPar Parser

- From single words:
  - stack.tag stack.word
  - input.tag input.word
  - input(1).tag input(1).word
  - input(1).tag input(1).word
  - input(2).tag input(2).word
  - input(2).tag input(2).word

- From word pairs:
  - stack.tag stack.word input.tag input.word
  - stack.tag stack.word input.tag input.word
  - stack.tag stack.word input.tag input.word
  - stack.tag stack.word input.tag input.word
  - stack.tag stack.word input.tag input.word
  - stack.tag stack.word input.tag input.word
  - stack.tag stack.word input.tag input.word

- From word triples:
  - input.tag input(1).tag input(2).tag
  - stack.tag input.tag input(1).tag
  - stack.head(1).tag stack.tag input.tag
  - stack.tag stack.child(-1).tag input.tag
  - stack.tag stack.child(1).tag input.tag
  - stack.tag input.tag input.child(-1).tag

- Distance:
  - stack.distance stack.word
  - stack.distance stack.tag
  - stack.distance input.word
  - stack.distance input.tag
  - stack.distance stack.word input.word
  - stack.distance stack.tag input.tag

- Valency:
  - stack.word stack.valence(-1)
  - stack.word stack.valence(1)
  - stack.tag stack.valence(-1)
  - stack.tag stack.valence(1)
  - input.word input.valence(-1)
  - input.tag input.valence(-1)

- Unigrams:
  - stack.head(1) {word tag}
  - stack.label
  - stack.child(-1) {word tag label}
  - stack.child(1) {word tag label}
  - input.child(-1) {word tag label}
  - stack.head(1).head(1) {word tag}
  - stack.head(1).label
  - stack.child(-1).sibling(1) {word tag label}
  - stack.child(1).sibling(-1) {word tag label}
  - input.child(-1).sibling(1) {word tag label}

- Third order:
  - stack.tag stack.child(-1).tag stack.child(-1).sibling(1).tag
  - stack.tag stack.child(1).tag stack.child(1).sibling(-1).tag
  - stack.tag stack.head(1).tag stack.head(1).head(1).tag
  - input.tag input.child(-1).tag input.child(-1).sibling(1).tag

- Label set:
  - stack.tag stack.child(-1).label
  - stack.tag stack.child(-1).label stack.child(-1).sibling(1).tag
  - stack.tag stack.child(-1).label stack.child(-1).sibling(1).sibling(2).tag
  - stack.tag stack.child(1).label
  - stack.tag stack.child(1).label stack.child(1).sibling(-1).label
  - stack.tag stack.child(1).label stack.child(1).sibling(-1).label stack.child(1).sibling(-2).label
  - input.tag input.child(-1).label
  - input.tag input.child(-1).label input.child(-1).sibling(1).label
  - input.tag input.child(-1).label input.child(-1).sibling(1).sibling(2).label
  - input.tag input.child(-1).label input.child(-1).sibling(1).sibling(2).sibling(3).label

Styles of Dependency Parsing

- Transition-Based (tr)
  - Fast, greedy, linear time inference algorithms
  - Trained for greedy search
  - Beam search

- Graph-Based (gr)
  - Slower, exhaustive, dynamic programming inference algorithms
  - Higher-order factorizations

Neural Network Transition Based Parser

[Chen & Manning '14] and [Weiss et al. '15]
Neural Network Transition Based Parser

[Weiss et al. '15]

![Diagram of Neural Network Transition Based Parser]

- **Embedding Layer**: Takes atomic inputs (words, POS, labels) and maps them to vectors.
- **Hidden Layers**: Process these vectors through multiple layers to capture complex patterns.
- **Softmax Layer**: Outputs probabilities of different parse tree structures.
- **Structured Perceptron**: Optional layer for refining the output of the softmax layer.

This parser is inspired by the work of Weiss et al. (2015) and is designed to parse natural language sentences efficiently.
**NN Hyperparameters**

- Regularization
- Loss function
- Dimensions
- Activation function
- Initialization
- Adagrad
- Dropout
- Mini-batch size
- Initial learning rate
- Learning rate schedule
- Momentum
- Stopping time
- Parameter averaging

**Effect of Embedding Dimensions**

Word Tuning on WSJ (Tune Set, \(D_{\text{pos}}=32\))

<table>
<thead>
<tr>
<th>Word Embedding Dimension (D_{\text{words}})</th>
<th>UAS (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>89.5</td>
</tr>
<tr>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>90.5</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
</tr>
<tr>
<td>16</td>
<td>91.5</td>
</tr>
<tr>
<td>32</td>
<td>92</td>
</tr>
<tr>
<td>64</td>
<td>92</td>
</tr>
<tr>
<td>128</td>
<td>92</td>
</tr>
</tbody>
</table>

- Pretrained 200x200
- Pretrained 200
- 200x200
- 200

**Tri-Training**

[Zhou et al. '05, Li et al. '14]

- ZPar Parser: UAS 96.35, LAS 95.02
- Berkeley Parser: UAS 89.84, LAS 87.21
- Agreement: ~40%

**Optimization matters!**

Use random restarts, grid
Pick best using holdout data

- **Tune**: WSJ S24
- **Dev**: WSJ S22
- **Test**: WSJ S23
**English Results (WSJ 23)**

<table>
<thead>
<tr>
<th>Method</th>
<th>UAS</th>
<th>LAS</th>
<th>Beam</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd-order Graph-based (ZM2014)</td>
<td>93.22</td>
<td>91.02</td>
<td></td>
</tr>
<tr>
<td>Transition-based Linear (ZN2011)</td>
<td>93.00</td>
<td>90.95</td>
<td>32</td>
</tr>
<tr>
<td>NN Baseline (Chen &amp; Manning, 2014)</td>
<td>91.80</td>
<td>89.60</td>
<td>1</td>
</tr>
<tr>
<td>NN Better SGD (Weiss et al., 2015)</td>
<td>92.58</td>
<td>90.54</td>
<td>1</td>
</tr>
<tr>
<td>NN Deeper Network (Weiss et al., 2015)</td>
<td>93.19</td>
<td>91.18</td>
<td>1</td>
</tr>
<tr>
<td>NN Perceptron (Weiss et al., 2015)</td>
<td>93.99</td>
<td>92.05</td>
<td>8</td>
</tr>
<tr>
<td>NN Semi-supervised (Weiss et al., 2015)</td>
<td>94.26</td>
<td>92.41</td>
<td>8</td>
</tr>
<tr>
<td>S-LSTM (Dyer et al., 2015)</td>
<td>93.20</td>
<td>90.90</td>
<td>1</td>
</tr>
<tr>
<td>Contrastive NN (Zhou et al., 2015)</td>
<td>92.83</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

**Tri-Training Impact**

- **WSJ §22 (Dev)**
  - **NN model benefits more from additional data**
  - **ZN does not improve even when using an alternative hyper graph model for Tri-training**

**English Out-of-Domain Results**

- Training on WSJ+Questions+Web
  - **Supervised**
  - **Semi-Supervised (Tri-Training)**

**Multilingual Results**

- **3rd-Order Graph (ZM2014)**
- **Transition-based Linear (ZN2011)**
- **Transition-based NN (B=32)**

*No tri-training data*

*With morph features*

[Alberti et al., EMNLP 2015]
Universal Dependencies

Stanford Universal++ Dependencies

Interset++ Morphological Features

Google Universal++ POS Tags

http://universaldependencies.github.io/docs/

Summary

- Dependency Parsing
  - Graph-based
  - Eisner Algorithm
  - Maximum Spanning Tree Algorithm
  - 1-Endpoint Crossing Algorithm
  - Transition Based Parsing
  - Neural Network Representations
  - Semi-Supervised Approaches
  - Data Representations