Syntax and Parsing II

Dependency Parsing

Slav Petrov – Google

Thanks to:
Dan Klein, Ryan McDonald, Alexander Rush, Joakim Nivre, Greg Durrett, David Weiss

Lisbon Machine Learning School 2015
Notes for 2016

- Can add 10min of material
Dependency Parsing

They solved the problem with statistics
(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):

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

- For a dependency graph $G = (V, A)$
- With label set $L = \{l_1, \ldots, l_{|L|}\}$

- $G$ is (weakly) connected:
  - If $i, j \in V$, $i \leftrightarrow^* j$.

- $G$ is acyclic:
  - If $i \rightarrow j$, then not $j \rightarrow^* i$.

- $G$ obeys the single-head constraint:
  - If $i \rightarrow j$, then not $i' \rightarrow j$, for any $i' \neq i$.

- $G$ is projective:
  - If $i \rightarrow j$, then $i \rightarrow^* i'$, for any $i'$ such that $i < i' < j$ or $j < i' < i$. 
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

<table>
<thead>
<tr>
<th>Time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(n)</td>
<td>greedy tr</td>
</tr>
<tr>
<td>O(k \cdot n)</td>
<td>k-best tr</td>
</tr>
<tr>
<td>O(n^3)</td>
<td>1st-order gr</td>
</tr>
<tr>
<td>O(n^3)</td>
<td>2nd-order gr</td>
</tr>
<tr>
<td>O(n^4)</td>
<td>3rd-order gr</td>
</tr>
</tbody>
</table>

[Nivre et al. ’03–’11] [McDonald et al. ’05–’06]
Arc-Factored Models

Assumes that the score/probability/weight of a dependency graph factors by its arcs

$$w(G) = \prod_{(i,j,k) \in G} w_{ij}^k$$

look familiar?

- $w_{ij}^k$ is the weight of creating a dependency from word $w_i$ to $w_j$ with label $l_k$

- Thus there is an assumption that each dependency decision is independent
  - Strong assumption! Will address this later.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
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: \(\max_{i,j,k} w(A) \times w(B) \times w_{hh'}^k\)

- 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
As McGwire neared, fans went wild

Eisner First-Order Parsing
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild.
As McGwire neared, fans went wild
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
Initialization: \( C[s][s][d][c] = 0.0 \quad \forall s, d, c \)

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

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

% Second: create complete items
\[
C[s][t][\leftarrow][1] = \max_{s \leq r < t} (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0])
\]
\[
C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1])
\]
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!
$x = \text{root John saw Mary}$
- Find highest scoring incoming arc for each vertex

- 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
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
  - root $\rightarrow$ saw $\rightarrow$ John is 40 (***)
  - root $\rightarrow$ John $\rightarrow$ 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.

Therefore, recursively call algorithm on new graph.
This is a tree and the MST for the contracted graph!!

Go back up recursive call and reconstruct final graph
Chu-Liu-Edmonds PseudoCode

Chu-Liu-Edmonds \((G_x, w)\)

1. Let \( M = \{(i^*, j) : j \in V_x, i^* = \arg \max_{i'} w_{ij}\} \)
2. Let \( G_M = (V_x, M) \)
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 \( < G_C, c, ma > = \) contract\((G, C, w)\)
6. Let \( G = \) Chu-Liu-Edmonds\((G_C, w)\)
7. Find vertex \( i \in C \) such that \((i', c) \in G \) and \( ma(i', c) = i \)
8. Find arc \((i'', i) \in C \)
9. Find all arc \((c, i''') \in G \)
10. \( G = G \cup \{(ma(c, i'''), i''')\}_{(c, i''') \in G} \cup C \cup \{(i', i)\} - \{(i'', i)\} \)
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**

```plaintext
contract(G = (V, A), C, w)
1. Let GC be the subgraph of G excluding nodes in C
2. Add a node c to GC representing cycle C
3. For i ∈ V − C : ∃ i′ ∈ C (i′, i) ∈ A
   Add arc (c, i) to GC 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 ∈ V − C : ∃ i′ ∈ C (i, i′) ∈ A
   Add edge (i, c) to GC with
   \[
   ma(i, c) = \arg \max_{i′ \in C} [\text{score}(i, i′) − \text{score}(a(i′), i′)]
   \]
   \[
   i′ = ma(i, c)
   \]
   \[
   \text{score}(i, c) = [\text{score}(i, i′) − \text{score}(a(i′), i′) + \text{score}(C)]
   \]
   \[
   \quad \text{where } a(v) \text{ is the predecessor of } v \text{ in } C
   \]
   \[
   \text{and } \text{score}(C) = \sum_{v\in C} \text{score}(a(v), v)
   \]
5. return < GC, c, ma >
```
Arc Weights

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

- 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

As McGwire neared, fans went wild.
(Structured) Perceptron

Training data: \( \mathcal{T} = \{(x_t, G_t)\}_{t=1}^{\mathcal{T}} \)

1. \( \mathbf{w}^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..\mathcal{T} \)
4. Let \( G' = \arg \max_{G'} \mathbf{w}^{(i)} \cdot f(G') \)
5. if \( G' \neq G_t \)
6. \[ \mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} + f(G_t) - f(G') \]
7. \( i = i + 1 \)
8. return \( \mathbf{w}^i \)
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: (σ,[i|β],A) ⇒ ([σ|i],β,A)
left-arc (label): ([σ|i|j],B,A) ⇒ ([σ|j],B,A∪{j,l,i})
right-arc (label): ([σ|i|j],B,A) ⇒ ([σ|i],B,A∪{i,l,j})
I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon

↑ Stack

booked

I

← Buffer

a flight to Lisbon

LEFT-ARC

nsubj

I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon

↑ Stack

a

I booked

← Buffer

flight to Lisbon

SHIFT

nsubj

I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon
Arc-Standard Example

↑ Stack

- a flight
- I booked

← Buffer

- to Lisbon

SHIFT

I booked a flight to Lisbon
Arc-Standard Example

↑ Stack

<table>
<thead>
<tr>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>a  flight</td>
</tr>
<tr>
<td>I  booked</td>
</tr>
</tbody>
</table>

← Buffer

| Lisbon |

SHIFT

I booked a flight to Lisbon
Arc-Standard Example

I booked a flight to Lisbon

↑ Stack

Lisbon

to

a flight

I booked

← Buffer

RIGHT-ARC

pobj

I booked a flight to Lisbon
I booked a flight to Lisbon.
Arc-Standard Example

I booked a flight to Lisbon

RIGHT-ARC
dobj

nsubj
det
prep
pobj

I booked a flight to Lisbon
Arc-Standard Example

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”
....
SVM / Structured Perceptron Hyperparameters

- Regularization
- Loss function
- **Hand-crafted features**
Features ZPar Parser

# From Single Words
pair { stack.tag stack.word }
stack { word tag }
pair { input.tag input.word }
input { word tag }
pair { input(1).tag input(1).word }
input(1) { word tag }
pair { input(2).tag input(2).word }
input(2) { word tag }

# From word pairs
quad { stack.tag stack.word input.tag input.word }
triple { stack.tag stack.word input.word }
triple { stack.tag stack.word input.tag }
triple { stack.word input.tag input.word }
triple { stack.tag input.tag input.word }
pair { stack.word input.word }
pair { stack.tag input.tag }
pair { input.tag input(1).tag }

# From word triples
triple { input.tag input(1).tag input(2).tag }
triple { stack.tag input.tag input(1).tag }
triple { stack.head(1).tag stack.tag input.tag }
triple { stack.tag stack.child(-1).tag input.tag }
triple { stack.tag stack.child(1).tag input.tag }
triple { stack.tag input.tag input.child(-1).tag }
triple { stack.tag stack.child(-1).tag stack.child(-1).sibling(1).tag }
triple { stack.tag stack.child(1).tag stack.child(1).sibling(-1).tag }
triple { stack.tag stack.head(1).tag stack.head(1).head(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(2).tag }

# Distance
pair { stack.distance stack.word }
pair { stack.distance stack.tag }
pair { stack.distance input.word }
pair { stack.distance input.tag }
triple { stack.distance stack.word input.word }
triple { stack.distance stack.tag input.tag }

# valency
pair { stack.word stack.valence(-1) }
pair { stack.word stack.valence(1) }
pair { stack.tag stack.valence(-1) }
pair { stack.tag stack.valence(1) }
pair { input.word input.valence(-1) }
pair { 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}

# third order
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}
triple { stack.tag stack.child(-1).tag stack.child(-1).sibling(1).tag }
triple { stack.tag stack.child(1).tag stack.child(1).sibling(-1).tag }
triple { stack.tag stack.head(1).tag stack.head(1).head(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(2).tag }

# label set
pair { stack.tag stack.child(-1).label }
quad { stack.tag stack.child(-1).label stack.child(-1).sibling(1).label }
quad { stack.tag stack.child(-1).label stack.child(-1).sibling(2).label }
quad { stack.tag stack.child(1).label stack.child(1).sibling(-1).label }
quad { stack.tag stack.child(1).label stack.child(1).sibling(-2).label }
quad { input.tag input.child(-1).label input.child(-1).sibling(1).label }
quad { input.tag input.child(-1).label input.child(-1).sibling(2).label }
quad { input.tag input.child(-1).label input.child(-1).sibling(3).label }
Neural Network Transition Based Parser

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

[Weiss et al. ’15]
Neural Network Transition Based Parser

[Weiss et al. ‘15]
Neural Network Transition Based Parser

[Weiss et al. ’15]

Softmax

Hidden Layer 2

Hidden Layer 1

Embedding Layer

Atomic Inputs
Neural Network Transition Based Parser

[Weiss et al. ‘15]
NN Hyperparameters

- Regularization
- Loss function
NN Hyperparameters

- Regularization
- Loss function
- Dimensions
- Activation function
- Initialization
- Adagrad
- Dropout
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
NN Hyperparameters

Optimization matters!
Use random restarts, grid
Pick best using holdout data

Tune: WSJ S24
Dev: WSJ S22
Test: WSJ S23
Random Restarts: How much Variance?

Variance of Networks on Tuning/Dev Set

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

2nd hidden layer + pre training increases correlation
Effect of Embedding Dimensions

Word Tuning on WSJ (Tune Set, $D_{pos, D_{labels}} = 32$)

<table>
<thead>
<tr>
<th>Word Tuning on WSJ (Tune Set, $D_{pos, D_{labels}} = 32$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretrained 200x200</td>
</tr>
<tr>
<td>Pretrained 200</td>
</tr>
<tr>
<td>200x200</td>
</tr>
<tr>
<td>200</td>
</tr>
</tbody>
</table>

UAS (%) vs Word Embedding Dimension ($D_{words}$)
Effect of Embedding Dimensions

POS/Label Tuning on WSJ (Tune Set, $D_{\text{words}} = 64$)

UAS (%) vs. POS/Label Embedding Dimension ($D_{\text{pos}}, D_{\text{labels}}$)

- Pretrained 200x200
- Pretrained 200
- 200x200
- 200
Tri-Training

[Zhou et al. ’05, Li et al. ’14]

ZPar Parser

UAS 89.96
LAS 87.26

Berkeley Parser

UAS 96.35
LAS 95.02

~40% agreement

UAS 89.84
LAS 87.21

~40% agreement
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**

Model

- Linear (ZN2011)
- NN (B=1)
- NN (B=8)
# 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>
English Out-of-Domain Results

- Train on WSJ + Web Treebank + QuestionBank
- Evaluate on Web

![Graph showing UAS results for different methods: 3rd Order Graph (ZM2014), Transition-based Linear (ZN 2011, B=32), Transition-based NN (B=1), Transition-based NN (B=8). The supervised and semi-supervised methods are compared with UAS values ranging from 87% to 90%.](image-url)
Multilingual Results

<table>
<thead>
<tr>
<th>Language</th>
<th>3rd-Order Graph (ZM2014)</th>
<th>Transition-based Linear (ZN2011)</th>
<th>Tensor-based Graph Lei et al. (2014)</th>
<th>Transition-based NN (B=32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalan</td>
<td>84</td>
<td>88.5</td>
<td>93</td>
<td>No tri-training data</td>
</tr>
<tr>
<td>Chinese</td>
<td>79.5</td>
<td>84</td>
<td>88.5</td>
<td>With morph features</td>
</tr>
<tr>
<td>Czech</td>
<td>75</td>
<td>79.5</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>75</td>
<td>79.5</td>
<td>88.5</td>
<td></td>
</tr>
<tr>
<td>German</td>
<td>88.5</td>
<td>93</td>
<td>88.5</td>
<td></td>
</tr>
<tr>
<td>Japanese</td>
<td>93</td>
<td>93</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>84</td>
<td>88.5</td>
<td>93</td>
<td></td>
</tr>
</tbody>
</table>

[Alberti et al., in submission]
Summary

- **Constituency Parsing**
  - CKY Algorithm
  - Lexicalized Grammars
  - Latent Variable Grammars
  - Conditional Random Field Parsing
  - Neural Network Representations

- **Dependency Parsing**
  - Eisner Algorithm
  - Maximum Spanning Tree Algorithm
  - Transition Based Parsing
  - Neural Network Representations