Syntax and Parsing II

Dependency Parsing

Slav Petrov – Google

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

Lisbon Machine Learning School 2018
Theysolvedtheproblemwithstatistics
(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):

```
(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>
<th>1st-order gr</th>
<th>2nd-order gr</th>
<th>3rd-order gr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$O(n)$</td>
<td>$O(n^3)$</td>
<td>$O(n^3)$</td>
<td>$O(n^4)$</td>
</tr>
</tbody>
</table>

- $O(\cdot \cdot)$
- $O(k \cdot n)$
- $O(n^k)$

[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

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

- \( w_{ij}^k \) 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.

Heads

Modifiers

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.
Dependency Representation

As McGwire neared, fans went wild.
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.
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.

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.

* 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
Eisner Algorithm Pseudo Code

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
  \( C'[s][t][\leftarrow][0] = \max_{s \leq r < t} \left( C'[s][r][\rightarrow][1] + C'[r + 1][t][\leftarrow][1] + s(t, s) \right) \)
  \( C'[s][t][\rightarrow][0] = \max_{s \leq r < t} \left( C'[s][r][\rightarrow][1] + C'[r + 1][t][\leftarrow][1] + s(s, t) \right) \)

  \% Second: create complete items
  \( C'[s][t][\leftarrow][1] = \max_{s \leq r < t} \left( C'[s][r][\leftarrow][1] + C'[r][t][\leftarrow][0] \right) \)
  \( C'[s][t][\rightarrow][1] = \max_{s < r \leq t} \left( C'[s][r][\rightarrow][0] + C'[r][t][\rightarrow][1] \right) \)

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 = root John saw Mary
Chu-Liu-Edmonds

- Find highest scoring incoming arc for each vertex

```
root
```
```
John 30 saw 20
```
```
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
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
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 $< G_C, c, ma > = \text{contract} (G, C, w)$
6. Let $G = \text{Chu-Liu-Edmonds} (G_C, w)$
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. $G = G \cup \{ (ma(c, i'''), i''') \} \cup G \cup C \cup \{ (i', i) \} - \{ (i'', i) \}$
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

\[ \text{contract}(G = (V, A), C, w) \]

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 \( i \in V - C : \exists i' \in C \) \( (i', i) \in A \)
   Add arc \((c, i)\) to \( G_C \) with
   \[ \text{ma}(c, i) = \arg \max_{i' \in C} \text{score}(i', i) \]
   \[ i' = \text{ma}(c, i) \]
   \[ \text{score}(c, i) = \text{score}(i', i) \]
4. For \( i \in V - C : \exists i' \in C \) \( (i, i'') \in A \)
   Add edge \((i, c)\) to \( G_C \) with
   \[ \text{ma}(i, c) = \arg \max_{i' \in C} [\text{score}(i, i') - \text{score}(a(i'), i')] \]
   \[ i' = \text{ma}(i, c) \]
   \[ \text{score}(i, c) = [\text{score}(i, i') - \text{score}(a(i'), i') + \text{score}(C)] \]
   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 \( < G_C, c, \text{ma} > \)
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
* As McGwire neared, fans went wild
Training data: \( T = \{(x_t, G_t)\}_{t=1}^{\left| T \right|} \)

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 \( w^i \)
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): \(([\sigma|i|j], B, A) \Rightarrow ([\sigma|j], B, A \cup \{j, l, i\})\)

right-arc (label): \(([\sigma|i|j], B, A) \Rightarrow ([\sigma|i], B, A \cup \{i, l, j\})\)
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
Arc-Standard Example

I booked a flight to Lisbon

↑ Stack

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

Buffer

SHIFT

a

flight

I

booked

to

Lisbon

I

booked

a

flight

to

Lisbon
Arc-Standard Example

↑ Stack

- to
- a flight
- I booked

← Buffer

- Lisbon

SHIFT

nsubj

I booked a flight to Lisbon

det
Arc-Standard Example

↑ Stack

Lisbon

to

a flight

I booked

LEFT-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
I booked a flight to Lisbon
I booked a flight to Lisbon.
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.word input.tag input.word }
triple { stack.tag stack.word input.tag }
triple { stack.tag input.tag input.word }
pair { stack.word input.word }
pair { stack.tag input.tag }
pair { input.tag input(1).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 }
triple { stack.distance input.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 }

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

[Chen & Manning ’14] and [Weiss et al. ’15, Andor et al. ’16]
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]

Embedding Layer

Hidden Layer 1

Hidden Layer 2

Softmax

Atomic Inputs

words

pos

labels
Neural Network Transition Based Parser

[Weiss et al. ‘15]
[Andor et al. ‘16]

structured perceptron

globally-normalized CRF
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, $D_{\text{pos}}$, $D_{\text{labels}}$ = 32)

UAS (%) vs. Word Embedding Dimension ($D_{\text{words}}$)

- Pretrained 200x200
- Pretrained 200
- 200x200
- 200
Effect of Embedding Dimensions

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

<table>
<thead>
<tr>
<th>UAS (%)</th>
<th>POS/Label Embedding Dimension ($D_{\text{pos}}, D_{\text{labels}}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.5</td>
<td>1</td>
</tr>
<tr>
<td>91.7</td>
<td>2</td>
</tr>
<tr>
<td>91.9</td>
<td>4</td>
</tr>
<tr>
<td>92.1</td>
<td>8</td>
</tr>
<tr>
<td>92.3</td>
<td>16</td>
</tr>
<tr>
<td>92.5</td>
<td>32</td>
</tr>
</tbody>
</table>

- Pretrained 200x200
- Pretrained 200
- 200x200
- 200
Do we need structure?

[Dozat & Manning ’17]
Bi-Affine Parsing

- Biaffine self-attention layer to score all possible heads for each dependent $i$

$$s_i^{(arc)} H^{(arc\text{-}head)} W \oplus b \quad h_i^{(arc\text{-}dep)} \oplus 1$$

- Train with cross-entropy
- Apply a spanning tree algorithm at inference time

Note: This is just an affine layer with a linear transformation!

$$s_i = H^{(arc\text{-}head)} (Wh_i^{(arc\text{-}dep)} + b)$$
Self-Attention

- From the AP comes this story
- From the AP comes this story

Source

Target
## 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 Results (WSJ 23)

<table>
<thead>
<tr>
<th>Type</th>
<th>Model</th>
<th>PTB</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition</td>
<td>Ballesteros et al. (2016)</td>
<td>93.56</td>
<td>91.42</td>
</tr>
<tr>
<td></td>
<td>Andor et al. (2016)</td>
<td>94.61</td>
<td>92.79</td>
</tr>
<tr>
<td></td>
<td>Kuncoro et al. (2016)</td>
<td>95.8</td>
<td>94.6</td>
</tr>
<tr>
<td></td>
<td>K&amp;G (2016)</td>
<td>93.9</td>
<td>91.9</td>
</tr>
<tr>
<td></td>
<td>Cheng et al. (2016)</td>
<td>94.10</td>
<td>91.49</td>
</tr>
<tr>
<td></td>
<td>Hashimoto et al. (2016)</td>
<td>94.67</td>
<td>92.90</td>
</tr>
<tr>
<td></td>
<td>D&amp;M (2017)</td>
<td>95.74</td>
<td>94.08</td>
</tr>
</tbody>
</table>
## Multilingual Results

<table>
<thead>
<tr>
<th>Treebanks</th>
<th>UPOS</th>
<th>XPOS</th>
<th>UAS</th>
<th>LAS</th>
<th>CLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>All treebanks</td>
<td>93.09</td>
<td>82.27</td>
<td>81.30</td>
<td>76.30</td>
<td>72.57</td>
</tr>
<tr>
<td>Large treebanks</td>
<td>95.58</td>
<td>94.56</td>
<td>85.16</td>
<td>81.77</td>
<td>78.40</td>
</tr>
<tr>
<td>Parallel treebanks</td>
<td>88.25</td>
<td>30.66</td>
<td>80.17</td>
<td>73.73</td>
<td>69.88</td>
</tr>
<tr>
<td>Small treebanks</td>
<td>87.02</td>
<td>82.03</td>
<td>70.19</td>
<td>61.02</td>
<td>54.76</td>
</tr>
<tr>
<td>Surprise treebanks</td>
<td>–</td>
<td>–</td>
<td>54.47</td>
<td>40.57</td>
<td>37.41</td>
</tr>
<tr>
<td>System</td>
<td>UPOS</td>
<td>XPOS</td>
<td>UAS</td>
<td>LAS</td>
<td>CLAS</td>
</tr>
<tr>
<td>Dozat et al.</td>
<td>93.09</td>
<td>82.27</td>
<td>81.30</td>
<td>76.30</td>
<td>72.57</td>
</tr>
<tr>
<td>Björkelund et al.</td>
<td>91.98</td>
<td>64.84</td>
<td>79.90</td>
<td>74.42</td>
<td>70.18</td>
</tr>
<tr>
<td>Yu et al.</td>
<td>91.00</td>
<td>79.93</td>
<td>74.22</td>
<td>68.41</td>
<td>63.24</td>
</tr>
<tr>
<td>Shi et al.</td>
<td>90.88</td>
<td>79.80</td>
<td>80.35</td>
<td>75.00</td>
<td>70.91</td>
</tr>
</tbody>
</table>
On January 13, 2018, a false ballistic missile alert was issued via the Emergency Alert System and Commercial Mobile Alert System over television, radio, and cellphones in the U.S. state of Hawaii. The alert stated that there was an incoming ballistic missile threat to Hawaii, advised residents to seek shelter, and concluded "This is not a drill". The message was sent at 8:07 a.m. local time.
SRL Systems: Pipelined vs. BIO-based

Pipeline Systems
- sentence, predicate
- syntactic features
  - argument id.
  - candidate argument spans
  - labeling
- labeled arguments
  - ILP/DP
  - prediction

Punyakanok et al., 2008
Tackstrom et al., 2015
FitzGerald et al., 2015

BIO-based Systems
- sentence, predicate
- word-level features
  - Deep BiLSTM + CRF layer
  - BIO sequence
  - Viterbi
  - prediction

Collobert et al., 2011
Zhou and Xu, 2015
Wang et al., 2015

DeepSRL
- sentence, predicate
  - No global normalization
  - Deep BiLSTM
  - BIO sequence
  - Hard constraints
  - prediction

He et al., 2017
SRL as a BIO Tagging Problem

Input Sentence & Predicate

Many tourists visit Disney to meet their favorite cartoon characters

BIO Output


Span Output

ARG0 V ARG1 AM-PRP
Many tourists visit Disney to meet their favorite cartoon characters.
DeepSRL Architecture (Revisit)

Input sentence:
Many tourists visit Disney to meet their favorite cartoon characters

Target Predicate:
[D] [I] [B] [B]

Word & Pred. Embeddings
Highway BiLSTMs
Tagging Softmax
Output Labels

B-A0  I-A0  B-V  B-V  ... ...
LSGN Architecture: Overview

(1) Construct span representations for all $n^2$ spans!

No predicate input!
Many tourists visit Disney to meet their favorite cartoon characters.
End-to-End SRL Results

- More improvements on Brown (out-domain) & OntoNotes (with nominal predicates)
- With ELMo, over 3 points improvement over ensemble model!
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

• Semantic Role Labeling