Dependency Parsing

Instructor: Yoav Artzi

Slides adapted from Dan Klein, Luke Zettlemoyer, Chris Manning, and Dan Jurafsky, and David Weiss
Overview

• The parsing problem
• Methods
  – Transition-based parsing
• Evaluation
• Projectivity
Parse Trees

• Part-of-speech Tagging:
  – Word classes

• Parsing:
  – From words to phrases to sentences
  – Relations between words

• Two views
  – Dependency
  – Constituency
Dependency Parsing

• Dependency structure shows which words depend on (modify or are arguments of) which other words.

The boy put the tortoise on the rug
Constituency (Phrase Structure) Parsing

- Phrase structure organizes words into nested constituents
- Linguists can, and do, argue about details
- Lots of ambiguity

new art critics write reviews with computers
• Syntactic structure consists of:
  – Lexical items
  – Binary asymmetric relations \(\rightarrow\) dependencies

Dependencies are typed with name of grammatical relation
Dependency Structure

• Syntactic structure consists of:
  – Lexical items
  – Binary asymmetric relations ➔ dependencies

submitted

Bills

nsubjpass

Head (governor, superior, regent)

Modifier (dependent, inferior, subordinate)

Arrow from head to modifier (but can be reversed)
Dependency Structure

- Syntactic structure consists of:
  - Lexical items
  - Binary asymmetric relations ➔ dependencies

Dependencies form a tree
• Syntactic structure consists of:
  – Lexical items
  – Binary asymmetric relations → dependencies

Dependencies form a tree

Bills

<table>
<thead>
<tr>
<th>prep</th>
<th>on</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prep</th>
<th>Prep</th>
<th>Prep</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nsubjpass</th>
<th>Auxpass</th>
<th>Prep</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Root</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Submitted</th>
<th>by</th>
<th>Pobj</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Bills | Brownback

<table>
<thead>
<tr>
<th>Senator</th>
<th>Republican</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>of</td>
</tr>
<tr>
<td></td>
<td>Kansas</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Immigration</th>
<th>and</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ports</th>
<th>Prep</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CC</th>
<th>Conj</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prep</th>
<th>Prep</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s Parse

John saw Mary

He said that the boy who was wearing the blue shirt with the white pockets has left the building.
Methods for Dependency Parsing

• Dynamic programming
  – Eisner (1996): \( O(n^3) \)

• Graph algorithms
  – McDonald et al. (2005): score edges independently using classifier and use maximum spanning tree

• Constraint satisfaction
  – Start with all edges, eliminate based on hard constraints

• “Deterministic parsing”
  – Left-to-right, each choice is done with a classifier
Making Decisions

What are the sources of information for dependency parsing?
1. Bilexical affinities
   – [issues ↳ the] is plausible
2. Dependency distance
   – mostly with nearby words
3. Intervening material
   – Dependencies rarely span intervening verbs or punctuation
4. Valency of heads
   – How many dependents on which side are usual for a head?

ROOT Discussion of the outstanding issues was completed.
MaltParse (Nivre et al. 2008)

• Greedy transition-based parser
• Each decision: how to attach each word as we encounter it
  – If you are familiar: like shift-reduce parser
• Select each action with a classifier
• The parser has:
  – a stack $\sigma$, written with the top to the right
    • which starts with the ROOT symbol
  – a buffer $\beta$, written with the top to the left
    • which starts with the input sentence
  – a set of dependency arcs $A$
    • which starts off empty
  – a set of actions
Arc-standard Dependency Parsing

Start: \( \sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset \)

- **Shift** \( \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A \)
- **Left-Arc\(_r\)** \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\} \)
- **Right-Arc\(_r\)** \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\} \)

Finish: \( \beta = \emptyset \)

ROOT Joe likes Marry
Arc-standard Dependency Parsing

Start: $\sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset$

- Shift: $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
- Left-Arc: $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\}$
- Right-Arc: $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\}$

Finish: $\beta = \emptyset$

**Example:**

```
ROOT
Joe likes Marry
```

<table>
<thead>
<tr>
<th>Shift</th>
<th>[ROOT]</th>
<th>[Joe, likes, marry]</th>
<th>$\emptyset$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-Arc</td>
<td>[ROOT]</td>
<td>[likes, marry]</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, likes]</td>
<td>[marry]</td>
<td>${(\text{likes},\text{Joe})} = A_1$</td>
</tr>
<tr>
<td>Right-Arc</td>
<td>[ROOT]</td>
<td>[likes]</td>
<td>$A_1$</td>
</tr>
<tr>
<td>Right-Arc</td>
<td>[]</td>
<td>[ROOT]</td>
<td>$A_1 \cup {(\text{likes},\text{Marry})} = A_2$</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT]</td>
<td>[]</td>
<td>$A_2 \cup {(\text{ROOT, likes})} = A_3$</td>
</tr>
</tbody>
</table>

```

```
Arc-standard Dependency Parsing

Start: \( \sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n, A = \emptyset \)
- **Shift** \( \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A \)
- **Left-Arc** \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\} \)
- **Right-Arc** \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i,w_j)\} \)

Finish: \( \beta = \emptyset \)

ROOT Happy children like to play with their friends.
Arc-eager Dependency Parsing

Start: \( \sigma = [\text{ROOT}], \beta = w_1, \ldots, w_n \), \( A = \emptyset \)

- **Left-Arc** \( _r \)
  \( \sigma | w_i, w_j | \beta, A \rightarrow \sigma, w_j | \beta, A \cup \{r(w_j, w_i)\} \)
  - Precondition: \( r'(w_k, w_i) \notin A, w_i \neq \text{ROOT} \)

- **Right-Arc** \( _r \)
  \( \sigma | w_i, w_j | \beta, A \rightarrow \sigma | w_i | w_j, \beta, A \cup \{r(w_i, w_j)\} \)

- **Reduce**
  \( \sigma | w_i, \beta, A \rightarrow \sigma, \beta, A \)
  - Precondition: \( r'(w_k, w_i) \in A \)

- **Shift**
  \( \sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A \)

Finish: \( \beta = \emptyset \)

This is the common “arc-eager” variant: a head can immediately take a right dependent, before *its* dependents are found.
1. Left-Arc, $σ|w_i, w_j|β, A \rightarrow σ, w_j|β, A∪\{r(w_j, w_i)\}
   Precondition: $r(w_k, w_i) \notin A$, $w_i \neq$ ROOT
2. Right-Arc, $σ|w_i, w_j|β, A \rightarrow σ|w_j, w_i|β, A∪\{r(w_i, w_j)\}$
3. Reduce $σ|w_i, β, A \rightarrow σ, β, A$
   Precondition: $r(w_k, w_i) \in A$
4. Shift $σ, w_i|β, A \rightarrow σ|w_i, β, A$

ROOT Happy children like to play with their friends.
Happy children like to play with their friends.
You terminate as soon as the buffer is empty. Dependencies = A₉
MaltParser (Nivre et al. 2008)

• Selecting the next action:
  – Discriminative classifier (SVM, MaxEnt, etc.)
  – Untyped choices: 4
  – Typed choices: $|R| \times 2 + 2$

• Features: POS tags, word in stack, word in buffer, etc.

• Greedy $\rightarrow$ no search
  – But can easily do beam search

• Close to state of the art

• Linear time parser $\rightarrow$ very fast!
Parsing with Neural Networks
Chen and Manning (2014)

• Arc-standard Transitions
  – Shift
  – Left-Arc\(_r\)
  – Right-Arc\(_r\)

• Selecting the next actions:
  – Untyped choices: 3
  – Typed choices: |R| * 2 + 1
  – Neural network classifier

• With a few model improvements and very careful hyper-parameter tuning gives SOTA results
Parsing with Neural Networks
Chen and Manning (2014)

[Chen & Manning, 2014]

Softmax Layer

Hidden Layer

Embedding Layer
(words  labels  pos)

stack₀-word = “ticket”
buffer₀-word = “to”
stack₀-label = “det”
buffer₀-POS = “IN”
Hyperparameters?

- Regularization
- Loss function
Hyperparameters?

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

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

Slide from David Weiss
Hyperparameters?
Hyperparameters?

Use random restarts, grid search
Pick best using holdout data

Tune: WSJ S24 (grid search)
Dev: WSJ S22 (development)
Test: WSJ S23 (final results)
Evaluation

Acc = \frac{\# \text{ correct deps}}{\# \text{ of deps}}

UAS = \frac{4}{5} = 80\%
LAS = \frac{2}{5} = 40\%
Projectivity

- Dependencies from CFG trees with head rules must be projective
  - Crossing arcs are not allowed
- But: theory allows to account for displaced constituents → non-projective structures

Who did Bill buy the coffee from yesterday?
Projectivity

• Arc-eager transition system:
  – Can’t handle non-projectivity

• Possible directions:
  – Give up!
  – Post-processing
  – Add new transition types
  – Switch to a different algorithm
    • Graph-based parsers (e.g., MSTParser)