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
  – Constituency
  – Dependency
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]
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
Dependency Structure

- Syntactic structure consists of:
  - Lexical items
  - Binary asymmetric relations \(\rightarrow\) dependencies

Dependencies are typed with name of grammatical relation

**Diagram:**

```
Bills
  prep
  on
  pobj
  ports
  cc
  conj

submitted
  auxpass
  prep

were

by
  pobj

Brownback
  nn
  appos

Senator

Republican
  prep
  of
  pobj
  Kansas
```

**Text:**

Bills were submitted by Brownback of Republican Kansas and immigration by Republican Kansas.
Dependency Structure

• Syntactic structure consists of:
  – Lexical items
  – Binary asymmetric relations \( \rightarrow \) dependencies
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
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 (CKY-style)
  - Similar to lexicalized PCFG: $O(n^5)$
  - 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** $\sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\}$
- **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 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: \( \beta = \emptyset \)

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, Joe]</td>
<td>[likes, marry]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, likes]</td>
<td>[marry]</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>Right-Arc</td>
<td>[ROOT]</td>
<td>[likes]</td>
<td>( {\text{likes}, \text{Joe}} = A_1 )</td>
</tr>
<tr>
<td>Right-Arc</td>
<td>[]</td>
<td>[ROOT]</td>
<td>( A_1 )</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT]</td>
<td>[]</td>
<td>( A_1 \cup {\text{likes}, \text{Marry}} = A_2 )</td>
</tr>
<tr>
<td></td>
<td></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** \( \sigma, w_i|\beta, A \rightarrow \sigma\wedge w_i, \beta, A \)
- **Left-Arc\(_r\)** \( \sigma\wedge w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j, w_i)\} \)
- **Right-Arc\(_r\)** \( \sigma\wedge w_i, w_j|\beta, A \rightarrow \sigma, w_i|\beta, A \cup \{r(w_i, w_j)\} \)

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.
Arc-eager

ROOT  Happy children like to play with their friends.
Happy children like to play with their friends.
Arg-eager

ROOT Happy children like to play with their friends.

1. Left-Arc_r σ|w_i, w_j|β, A → σ, w_j|β, A ∪ {r(w_i, w_j)}
   Precondition: r'(w_k, w_i) ∉ A, w_i ≠ ROOT

2. Right-Arc_r σ|w_i, w_j|β, A → σ|w_i|w_j, β, A ∪ {r(w_i, w_j)}

3. Reduce σ|w_i, β, A → σ, β, A
   Precondition: r'(w_k, w_i) ∈ A

4. Shift σ, w_j|β, A → σ|w_i, β, A

You terminate as soon as the buffer is empty. Dependencies = A_9
MaltParser (Nivre et al. 2008)

- Selecting the next action:
  - Discriminative classifier (SVM, MaxEnt, etc.)
  - Untyped choices: 4
  - Typed choices: |R| * 2 + 2
- Features: POS tags, word in stack, word in buffer, etc.
- Greedy → no search
  - But can easily do beam search
- Close to state of the art
- Linear time parser → 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 training and model improvements 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”
Evaluation

Gold

1 2  She  nsubj
2 0  saw  root
3 5  the  det
4 5  video  nn
5 2  lecture  dobj

Parsed

1 2  She  nsubj
2 0  saw  root
3 4  the  det
4 5  video  nsubj
5 2  lecture  ccomp

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)