Dependency Parsing

Instructor: Yoav Artzi
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.
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

Bills were submitted by Brownback Republican Kansas of immigration and on ports

nsubjpass auxpass prep

Bills

on

pobj

port

cc conj

and

immigration

submitted

were

by

pobj

Brownback

nn appos

Senator

prep

of

pobj

Kansas
• Syntactic structure consists of:
  – Lexical items
  – Binary asymmetric relations \( \rightarrow \) dependencies

Dependencies form a tree

- \text{Root}
- \text{submitted}
- \text{were}
- \text{by}
- \text{Brownback}
- \text{Republican}
- \text{Senator}
- \text{Kansas}
- \text{ports}
- \text{immigration}
- \text{and}
- \text{cc}
- \text{conj}
- \text{pobj}
- \text{pobj}
- \text{pobj}
- \text{pobj}
- \text{prep}
- \text{nn}
- \text{appos}
- \text{appos}
- \text{auxpass}
- \text{nsbjpass}
- \text{prep}
- \text{on}
- \text{pobj}
- \text{prep}
- \text{prep}
- \text{prep}
He said that the boy who was wearing the blue shirt with the white pockets has left the building.

John saw Mary.
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\(_r\)**
  \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\} \)

- **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

<table>
<thead>
<tr>
<th>Action</th>
<th>Stack</th>
<th>Buffer</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift</td>
<td>[ROOT]</td>
<td>[Joe, likes, marry]</td>
<td>\emptyset</td>
</tr>
<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>A(_1)</td>
</tr>
<tr>
<td>Right-Arc</td>
<td>[ROOT]</td>
<td>[likes]</td>
<td>A(_1) u {(likes,Marry)} = A(_2)</td>
</tr>
<tr>
<td>Right-Arc</td>
<td>[]</td>
<td>[ROOT]</td>
<td>A(_2) u {(ROOT, likes)} = A(_3)</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT]</td>
<td>[]</td>
<td>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\(_r\)** \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, A \cup \{r(w_j,w_i)\} \)
- **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** 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

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

ROOT Happy children like to play with their friends.

1. Left-Arc, \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma, w_j|\beta, \text{Au}\{r(w_i, w_j)\} \)
   Precondition: \( r(w_i, w_j) \notin A, w_j \neq \text{ROOT} \)

2. Right-Arc, \( \sigma|w_i, w_j|\beta, A \rightarrow \sigma|w_i, w_j, \text{Au}\{r(w_i, w_j)\} \)

3. Reduce, \( \sigma|w_i, \beta, A \rightarrow \sigma, \beta, A \)
   Precondition: \( r(w_i, w_j) \in A \)

4. Shift, \( \sigma, w_j|\beta, A \rightarrow \sigma|w_i, \beta, A \)

<table>
<thead>
<tr>
<th>Action</th>
<th>Stack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT</td>
<td>[ROOT]</td>
<td>[Happy, children, ...]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, Happy]</td>
<td>[children, like, ...]</td>
</tr>
<tr>
<td>LA_amod</td>
<td>[ROOT]</td>
<td>[children, like, ...]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, children]</td>
<td>[like, to, ...]</td>
</tr>
<tr>
<td>LA_nsubj</td>
<td>[ROOT]</td>
<td>[like, to, ...]</td>
</tr>
<tr>
<td>RA_root</td>
<td>[ROOT, like]</td>
<td>[to, play, ...]</td>
</tr>
<tr>
<td>Shift</td>
<td>[ROOT, like, to]</td>
<td>[play, with, ...]</td>
</tr>
<tr>
<td>LA_aux</td>
<td>[ROOT, like]</td>
<td>[play, with, ...]</td>
</tr>
<tr>
<td>RA_xcomp</td>
<td>[ROOT, like, play]</td>
<td>[with their, ...]</td>
</tr>
</tbody>
</table>
**Arc-eager**

ROOT **Happy children like to play with their friends.**

1. **Left-Arc**, $\sigma|w_i, w_j|\beta, A \Rightarrow \sigma, w_j|\beta, A\cup\{r(w_i, w_j)\}$
   
   **Precondition:** $r(w_i, w_j) \notin A$, $w_i \neq \text{ROOT}$

2. **Right-Arc**, $\sigma|w_i, w_j|\beta, A \Rightarrow \sigma|w_i|\beta, A\cup\{r(w_i, w_j)\}$

3. **Reduce**, $\sigma|w_i, \beta, A \Rightarrow \sigma, \beta, A$
   
   **Precondition:** $r(w_i, w_j) \in A$

4. **Shift**, $\sigma, w_i|\beta, A \Rightarrow \sigma|w_i, \beta, A$

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 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)

stack_0-word = “ticket”
buffer_0-word = “to”
stack_0-label = “det”
buffer_0-POS = “IN”
Hyperparameters?

- Regularization
- Loss function

Slide from David Weiss
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
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

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 = \# correct deps / \# of deps

UAS = 4 / 5 = 80%
LAS = 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)