CS395T: Structured Models for NLP
Lecture 9: Trees 3

Greg Durrett

Administrivia
- Project 1 due at *5pm* today
- Project 2 will be out by tonight. Due October 17
  - Shift-reduce parser: greedy model, beam search model, extension

Recall: Dependencies
- Dependency syntax: syntactic structure is defined by dependencies
- Head (parent, governor) connected to dependent (child, modifier)
- Each word has exactly one parent except for the ROOT symbol
- Dependencies must form a directed acyclic graph

Recall: Projectivity
- Projective <-> no “crossing” arcs
- Crossing arcs:
- Today: algorithms for projective parsing

信用：Language Log
This Lecture

- Graph-based dependency parsing
- Dynamic programs for exact inference — look a lot like sequential CRFs
- Transition-based (shift-reduce) dependency parsing
- Approximate, greedy inference — fast, but a little bit weird!

Graph-based Dependency Parsing

- How did we parse lexicalized trees?
- Normal CKY is too slow: grammar is too large if it includes words

Graph-based Dependency Parsing

- Naive algorithm: \(O(n^5)\)
- Combine spans like CKY and look at their heads
- Five indices to loop over
- Features can look at spans and heads
- Can be applied to dependency parses as well! Builds projective trees
- What do our scores look like? For now, assume features on edge (head, child) pair with some weights

Why is this inefficient?

- Lots of spurious ambiguity — many ways to derive the right parses
- Can split at either point and we can build up subtrees
Eisner’s Algorithm: $O(n^3)$

- Cubic-time algorithm like CKY
- Maintain two charts with dimension $[n, n, 2]$:
  - Complete items: all children are attached, head is at the “tall end”
  - Incomplete items: arc from “tall” to “short” end, word on short end has parent but maybe not all of its children

Complete items: all children are attached, head is at the “tall end”
Incomplete item: arc from “tall end” to “short end”, may still expect children

Take two adjacent complete items, add arc and build incomplete item

Take an incomplete item, complete it (other case is symmetric)
Eisner’s Algorithm: $O(n^3)$

- Attaching to ROOT makes an incomplete item with left children, attaches with right children subsequently to finish the parse
- We’ve built left children and right children of ran as complete items

Eisner’s Algorithm

- Eisner’s algorithm doesn’t have split point ambiguities like this
- Left and right children are built independently, heads are edges of spans
- Charts are $n \times n \times 2$ because we need to track arc direction / left vs right

MST Parser

- View dependency parsing as finding a maximum direct spanning tree — space of all spanning trees, so we find nonprojective trees too!
- Chu-Liu-Edmonds algorithm to find the best MST in $O(n^2)$
- This only computes maxes, but there is an algorithm for summing over all trees as well (matrix-tree theorem)
- Ironically, the software artifact called MST Parser has an implementation of Eisner’s algorithm, which is what most people use

Building Systems

- Can implement Viterbi decoding and marginal computation using Eisner’s algorithm or MST to max/sum over projective/nonprojective trees
- Same concept as sequential CRFs for NER, can also use margin-based methods — you know how to implement these!
- Features are over dependency edges

McDonald et al. (2005)
Features in Graph-Based Parsing

- Dynamic program exposes the parent and child indices
  
  ROOT DT NN VBD TO DT NN
  - McDonald et al. (2005) — conjunctions of parent and child words + POS, POS of words in between, POS of surrounding words. ~91 UAS
  - HEAD=TO & MOD=NN
  - HEAD=TO & MOD=house
  - HEAD=TO & MOD=the
  - HEAD=TO & MOD=DT
  - Lei et al. (2014) — ways of learning conjunctions of these

- Ideally would use features on more arcs
  
  ROOT DT NN VBD TO DT NN
  - Grandparents: ran -> to -> house
  - Siblings: dog <-> ran <-> to

Higher-Order Parsing

- Terry Koo (2010)
  
  Track additional state during parsing so we can look at grandparents and siblings, $O(n^4)$
  
  Additional indicator features based on this information, ~93 UAS (up from 91 UAS)
  
  Turns out you can just use beam search and forget this crazy dynamic program...

Shift-Reduce Parsing
Shift-Reduce Parsing

- Similar to deterministic parsers for compilers
- Also called transition-based parsing
- A tree is built from a sequence of incremental decisions moving left to right through the sentence
- Stack containing partially-built tree, buffer containing rest of sentence
- Shifts consume the buffer, reduces build a tree on the stack

Shift-Reduce Parsing

ROOT

I ate some spaghetti bolognese

- Initial state: Stack: [ROOT]  Buffer: [I ate some spaghetti bolognese]
- Shift: top of buffer -> top of stack
  - Shift 1: Stack: [ROOT I]  Buffer: [ate some spaghetti bolognese]
  - Shift 2: Stack: [ROOT I ate]  Buffer: [some spaghetti bolognese]

Shift-Reduce Parsing

ROOT

I ate some spaghetti bolognese

- Start: stack contains [ROOT], buffer contains [I ate some spaghetti bolognese]
- Arc-standard system: three operations
  - Shift: top of buffer -> top of stack
    - Left-Arc: \( \sigma[w_{-2}, w_{-1}] \rightarrow \sigma[w_{-1}] \), \( w_{-2} \) is now a child of \( w_{-1} \)
    - Right-Arc: \( \sigma[w_{-2}, w_{-1}] \rightarrow \sigma[w_{-2}] \), \( w_{-1} \) is now a child of \( w_{-2} \)
  - End: stack contains [ROOT], buffer is empty []
- Must take \( 2n \) steps for \( n \) words (\( n \) shifts, \( n \) LA/RA)
**Arc-Standard Parsing**

**ROOT**
I ate some spaghetti bolognese

[S top of buffer -> top of stack](LA pop two, left arc between them) [RA pop two, right arc between them]

- Could do the left arc later! But no reason to wait
- Can’t attach ROOT <- ate yet even though this is a correct dependency!

**Arc-Standard Parsing**

- [ROOT] S S [I ate some spaghetti bolognese] L
- [ROOT I] S S [I some spaghetti bolognese] L
- [ROOT I ate] L [some spaghetti bolognese] L
- [ROOT ate] L [some spaghetti bolognese] L

**Other Systems**

- Arc-eager (Nivre, 2004): lets you add right arcs sooner and keeps items on stack, separate reduce action that clears out the stack
- Arc-swift (Qi and Manning, 2017): explicitly choose a parent from what’s on the stack
- Many ways to decompose these, which one works best depends on the language and features
### Building Shift-Reduce Parsers

<table>
<thead>
<tr>
<th>[ROOT]</th>
<th>[I ate some spaghetti bolognese]</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do we make the right decision in this case?</td>
<td></td>
</tr>
<tr>
<td>Only one legal move (shift)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[ROOT ate some spaghetti]</th>
<th>[bolognese]</th>
</tr>
</thead>
<tbody>
<tr>
<td>How do we make the right decision in this case? (all three actions legal)</td>
<td></td>
</tr>
<tr>
<td>Correct action is left-arc</td>
<td></td>
</tr>
<tr>
<td>Multi-way classification problem: shift, left-arc, or right-arc?</td>
<td></td>
</tr>
</tbody>
</table>

### Features for Shift-Reduce Parsing

<table>
<thead>
<tr>
<th>[ROOT ate some spaghetti]</th>
<th>[bolognese]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features to know this should left-arc?</td>
<td></td>
</tr>
<tr>
<td>One of the harder feature design tasks!</td>
<td></td>
</tr>
<tr>
<td>In this case: the stack tag sequence VBD - DT - NN is pretty informative — looks like a verb taking a direct object which has a determiner in it</td>
<td></td>
</tr>
<tr>
<td>Things to look at: top words/POS of buffer, top words/POS of stack, leftmost and rightmost children of top items on the stack</td>
<td></td>
</tr>
</tbody>
</table>

### Training a Greedy Model

<table>
<thead>
<tr>
<th>[ROOT ate some spaghetti]</th>
<th>[bolognese]</th>
</tr>
</thead>
<tbody>
<tr>
<td>The algorithm we’ve developed so far is an oracle, tells us the correct state transition sequence for each tree</td>
<td></td>
</tr>
<tr>
<td>Use our oracle to extract parser states + correct decisions</td>
<td></td>
</tr>
<tr>
<td>Train a classifier to predict the right decision using these as training data</td>
<td></td>
</tr>
<tr>
<td>Problem: no look ahead</td>
<td></td>
</tr>
<tr>
<td>No lookahead</td>
<td></td>
</tr>
<tr>
<td>Training data is extracted assuming everything is correct</td>
<td></td>
</tr>
</tbody>
</table>

### Dynamic Oracle

<table>
<thead>
<tr>
<th>[ROOT ate some spaghetti]</th>
<th>[bolognese]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract training data based on the oracle but also an execution trace of a trained parser</td>
<td></td>
</tr>
<tr>
<td>Need a dynamic oracle to determine what’s the optimal thing to do even if mistakes have already been made (so we know how to supervise it)</td>
<td></td>
</tr>
<tr>
<td>We’ll see similar ideas in neural net contexts as well</td>
<td></td>
</tr>
</tbody>
</table>

Goldberg and Nivre (2012)
### Speed Tradeoffs

<table>
<thead>
<tr>
<th>Parser</th>
<th>Dev UAS</th>
<th>Dev LAS</th>
<th>Test UAS</th>
<th>Test LAS</th>
<th>Speed (sent/seg)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unoptimized S-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard</td>
<td>89.9</td>
<td>88.7</td>
<td>89.7</td>
<td>88.3</td>
<td>51</td>
</tr>
<tr>
<td>eager</td>
<td>90.3</td>
<td>89.2</td>
<td>89.9</td>
<td>88.6</td>
<td>63</td>
</tr>
<tr>
<td>Optimized S-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malt:sp</td>
<td>90.0</td>
<td>88.8</td>
<td>89.9</td>
<td>88.5</td>
<td>560</td>
</tr>
<tr>
<td>Malt:eager</td>
<td>90.1</td>
<td>88.9</td>
<td>90.1</td>
<td>88.7</td>
<td>535</td>
</tr>
<tr>
<td>Graph-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSTParser</td>
<td>92.1</td>
<td>90.8</td>
<td>92.0</td>
<td>90.5</td>
<td>12</td>
</tr>
<tr>
<td>Neural S-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our parser</td>
<td>92.2</td>
<td>91.0</td>
<td>92.0</td>
<td>90.7</td>
<td><strong>1013</strong></td>
</tr>
</tbody>
</table>

- Optimized constituency parsers are ~5 sentences/sec
- Using S-R used to mean taking a performance hit compared to graph-based, that’s no longer true

Chen and Manning (2014)

### Global Decoding

- [ROOT ate some spaghetti]  [bolognese]

  
  - Try to find the highest-scoring sequence of decisions
  - Global search problem, requires approximate search

---

**Global Decoding**

1. **Correct:** Right-arc, Shift, Right-arc, Right-arc

   - [ROOT gave]  [dinner]
   - [ROOT gave]  [dinner]
   - [ROOT gave dinner]  [dinner]
   - [ROOT gave]  [dinner]
   - [ROOT gave]  [dinner]

   ![Correct parse](image)

- Both wrong! Also both probably low scoring!

2. **Correct, high scoring option**

   - [ROOT gave]  [dinner]
   - [ROOT gave]  [dinner]
   - [ROOT gave]  [dinner]
   - [ROOT gave]  [dinner]
   - [ROOT gave]  [dinner]

   ![Correct parse](image)
## Global Decoding: A Cartoon

- Lookahead can help us avoid getting stuck in bad spots
- Global model: maximize sum of scores over all decisions
- Similar to how Viterbi works: we maintain uncertainty over the current state so that if another one looks more optimal going forward, we can use that one

## Recap

- Eisner’s algorithm for graph-based parsing
- Arc-standard system for transition-based parsing
- Run a classifier and do it greedily for now, we’ll see global systems next time