CS388: Natural Language Processing
Lecture 11: Dependency Parsing I

Greg Durrett
Project 1 graded by Tuesday

Survey results:
- Some annoyances from projects: slow debugging/training, etc.
  - If you have comments on the code, please send them to me (either anonymously or non-anonymously)
- Bit rate
- Clearer slides/notation
Recall: Constituency

- Tree-structured syntactic analyses of sentences
- Nonterminals (NP, VP, etc.) as well as POS tags (bottom layer)
- Structured is defined by a CFG
Recall: CKY

- Find $\text{argmax } P(T | x) = \text{argmax } P(T, x)$

- Dynamic programming: chart maintains the best way of building symbol $X$ over span $(i, j)$

- Loop over all split points $k$, apply rules $X \rightarrow Y Z$ to build $X$ in every possible way

Cocke-Kasami-Younger
Outline

- Discriminative constituency parsing
- Dependency representation, contrast with constituency
- Projectivity
- Graph-based dependency parsers
Discriminative Parsers
CRF Parsing

\[
\text{score}\left( \begin{array}{c}
\text{NP} \\
\text{NP} \\
\text{PP}
\end{array} \right) = \mathbf{w}^\top f\left( \begin{array}{c}
\text{NP} \\
\text{NP} \\
\text{PP}
\end{array} \right)
\]

He wrote a long report on Mars.

\[
f\left( \begin{array}{c}
\text{NP} \\
\text{NP} \\
\text{PP}
\end{array} \right) = \left[ \begin{array}{c}
\text{NP} \\
\text{NP} \\
\text{PP}
\end{array} \right]
\]

Left child last word = report \wedge \text{NP} \rightarrow [\text{PP}]

- Can learn that we report [PP], which is common due to reporting on things.
- Can “neuralize” this as well like neural CRFs for NER.

Taskar et al. (2004)
Hall, Durrett, and Klein (2014)
Durrett and Klein (2015)
Joint Discrete and Continuous Parsing

- Chart remains discrete!

Parsing a sentence:
- Feedforward pass on nets
- Discrete feature computation
- Run CKY dynamic program

He wrote a long report on Mars

Durrett and Klein (ACL 2015)
Simpler version: score *constituents* rather than rule applications

Use BiLSTMs to compute embeddings of each word, embeddings at edge of span characterize that span

91-93 F1, 95 F1 with ELMo (SOTA).
Great on other langs too!

He wrote a long report on Mars.

---

Stern et al. (2017), Kitaev et al. (2018)
Dependency Representation
Lexicalized Parsing

S(ran)
  /   \
VP(ran)    PP(to)
  /      |
NP(dog)   NP(house)
  /  \
DT(the) NN(dog)  VBD(ran) TO(to)
    \
the   dog ran  to

the
NN(house)

the

Dependency Parsing

- Dependency syntax: syntactic structure is defined by these arcs
- 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

ROOT

ROOT

POS tags same as before, usually run a tagger first as preprocessing
Still a notion of hierarchy! Subtrees often align with constituents
Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)
Constituency: several rule productions need to change
Dependency vs. Constituency: PP Attachment

- Dependency: one word (with) assigned a different parent

  the children ate the cake with a spoon

- More predicate-argument focused view of syntax

- “What’s the main verb of the sentence? What is its subject and object?” — easier to answer under dependency parsing
Dependency vs. Constituency: Coordination

- Constituency: ternary rule NP -> NP CC NP
Dependency vs. Constituency: Coordination

- Dependency: first item is the head

  dogs in houses and cats
  [dogs in houses] and cats
  dogs in houses and cats
  dogs in [houses and cats]

- Coordination is decomposed across a few arcs as opposed to being a single rule production as in constituency

- Can also choose and to be the head

- In both cases, headword doesn’t really represent the phrase — constituency representation makes more sense
Universal Dependencies

- Annotate dependencies with the same representation in many languages

---

**English**

1. The **det** dog **aux:pass** was **verb** chased **obl** by the **det** cat

---

**Bulgarian**

2. Кучeto **nsubj:pass** ce **expl:pass** преследваше **verb** от **obl** котката

---

**Czech**

3. Pes **nsubj:pass** byl **aux:pass** honěn **verb** kočkou **obl**

---

**Swiss**

4. Hunden **nsubj:pass** jagades **verb** av **obl** katten

---

http://universaldependencies.org/
Any subtree is a contiguous span of the sentence $\leftrightarrow$ tree is *projective*
Projectivity

- Projective $\leftrightarrow$ no “crossing” arcs

- Crossing arcs:

  dogs in houses and cats

  the dog ran to the house

credit: Language Log
Projectivity in other languages

- Swiss-German has famous non-context-free constructions

credit: Pitler et al. (2013)
Projectivity

- Number of trees produceable under different formalisms

<table>
<thead>
<tr>
<th></th>
<th>Arabic</th>
<th>Czech</th>
<th>Danish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projective</td>
<td>1297 (88.8)</td>
<td>55872 (76.8)</td>
<td>4379 (84.4)</td>
</tr>
<tr>
<td>Sentences</td>
<td>1460</td>
<td>72703</td>
<td>5190</td>
</tr>
</tbody>
</table>

- Many trees in other languages are nonprojective

Pitler et al. (2013)
### Projectivity

- Number of trees produceable under different formalisms

<table>
<thead>
<tr>
<th>Formalism</th>
<th>Arabic</th>
<th>Czech</th>
<th>Danish</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Endpoint-Crossing</td>
<td>1457 (99.8)</td>
<td>71810 (98.8)</td>
<td>5144 (99.1)</td>
</tr>
<tr>
<td>Well-nested, block degree 2</td>
<td>1458 (99.9)</td>
<td>72321 (99.5)</td>
<td>5175 (99.7)</td>
</tr>
<tr>
<td>Gap-Minding</td>
<td>1394 (95.5)</td>
<td>70695 (97.2)</td>
<td>4985 (96.1)</td>
</tr>
<tr>
<td>Projective</td>
<td>1297 (88.8)</td>
<td>55872 (76.8)</td>
<td>4379 (84.4)</td>
</tr>
<tr>
<td>Sentences</td>
<td>1460</td>
<td>72703</td>
<td>5190</td>
</tr>
</tbody>
</table>

- Many trees in other languages are nonprojective
- Some other formalisms (that are harder to parse in), most useful one is 1-Endpoint-Crossing

Pitler et al. (2013)
Graph-Based Parsing
Defining Dependency Graphs

- Words in sentence $\mathbf{x}$, tree $T$ is a collection of directed edges (parent($i$), $i$) for each word $i$
  - Parsing = identify parent($i$) for each word
  - Each word has exactly one parent. Edges must form a projective tree

- Log-linear CRF (discriminative): $P(T|\mathbf{x}) = \exp \left( \sum_{i} w^\top f(i, \text{parent}(i), \mathbf{x}) \right)$

- Example of a feature = $I[\text{head=to} \& \text{modifier=house}]$ (more in a few slides)

ROOT the dog ran to the house
Generalizing CKY

- Score matrix with three dimensions: start, end, and head, start <= head < end
- new score = score(2, 5, 4) + score(5, 7, 5) + edge score(4 -> 5)
- score(2, 7, 4) = max(score(2, 7, 4), new score)
- Time complexity of this?
- Many spurious derivations: can build the same tree in many ways...need a better algorithm
Eisner’s Algorithm: $O(n^3)$

- Cubic-time algorithm
- Maintain two dynamic programming charts with dimension $[n, n, 2]$:
  - Complete items: head is at “tall end”, may be missing children on tall side
  - Incomplete items: arc from “tall” to “short” end, word on short end may also be missing children
Eisner’s Algorithm: $O(n^3)$

- **Complete item**: 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

  \[ \begin{array}{c}
  \text{Complete item: all children are attached, head is at the “tall end”} \\
  \text{Incomplete item: arc from “tall end” to “short end”, may still expect children} \\
  \text{Take two adjacent complete items, add arc and build incomplete item} \\
  \text{Take an incomplete item, complete it} \\
  \text{(other case is symmetric)}
  \end{array} \]

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

  \[ \begin{array}{c}
  \text{Complete item: all children are attached, head is at the “tall end”} \\
  \text{Incomplete item: arc from “tall end” to “short end”, may still expect children} \\
  \text{Take two adjacent complete items, add arc and build incomplete item} \\
  \text{Take an incomplete item, complete it} \\
  \text{(other case is symmetric)}
  \end{array} \]

- Complete item: all children are attached, head is at the “tall end”
- Incomplete item: arc from “tall end” to “short end”, may still expect children
Eisner’s Algorithm: $O(n^3)$

1) Build incomplete span

2) Promote to complete

3) Build incomplete span

ROOT  DT  NN  VBD  TO  DT  NN
the  the  dog  ran  to  house
Eisner’s Algorithm: $O(n^3)$

4) Promote to complete

ROOT

DT the

NN dog

VBD ran

TO to

DT the

NN house
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 CKY does
- 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

Eisner:

$\text{ROOT} \rightarrow \text{DT the} \rightarrow \text{NN dog} \rightarrow \text{VBD ran} \rightarrow \text{TO to} \rightarrow \text{DT the} \rightarrow \text{NN house}$
Building Systems

- Can implement decoding and marginal computation using Eisner’s algorithm to max/sum over projective trees

- Conceptually the same as inference/learning for sequential CRFs for NER, can also use margin-based methods
Features in Graph-Based Parsing

- Dynamic program exposes the parent and child indices

  \[ f(i, \text{parent}(i), x) \]

- McDonald et al. (2005) — conjunctions of parent and child words + POS, POS of words in between, POS of surrounding words
  - HEAD=TO & MOD=NN
  - HEAD=TO & MOD-1=the
  - HEAD=TO & MOD=house
  - ARC_CROSSES=DT
Higher-Order Parsing

\[ f(i, \text{parent}(i), \text{parent(parent}(i)), x) \]

- Track additional state during parsing so we can look at “grandparents” (and siblings). \( O(n^4) \) dynamic program or use approximate search

Koo and Collins (2009)
Neural CRFs for dependency parsing: let $c = \text{LSTM embedding of } i$, $p = \text{LSTM embedding of parent}(i)$. $score(i, \text{parent}(i), x) = p^T U c$

LSTM looks at words and POS

Dozat and Manning (2017)
Evaluating Dependency Parsing

- **UAS**: unlabeled attachment score. Accuracy of choosing each word’s parent (n decisions per sentence)

- **LAS**: additionally consider label for each edge

- Log-linear CRF parser, decoding with Eisner algorithm: 91 UAS

- Higher-order features from Koo parser: 93 UAS

- Best English results with neural CRFs: 95-96 UAS
Takeaways

- Dependency formalism provides an alternative to constituency, particularly useful in how portable it is across languages.

- Dependency parsing also has efficient dynamic programs for inference.

- CRFs + neural CRFs (again) work well.