CS11-747 Neural Networks for NLP

Parsing with Dynamic Programming

Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/nn4nlp2017/
Two Types of Linguistic Structure

- **Dependency:** focus on relations between words
  
  ![Dependency Diagram]

- **Phrase structure:** focus on the structure of the sentence
  
  ![Phrase Structure Diagram]
Parsing

• Predicting linguistic structure from input sentence

• **Transition-based models**
  • step through actions one-by-one until we have output
  • like history-based model for POS tagging

• **Dynamic programming-based models**
  • calculate probability of each edge/constituent, and perform some sort of dynamic programming
  • like linear CRF model for POS
Minimum Spanning Tree
Parsing Models
(First Order) Graph-based Dependency Parsing

- Express sentence as fully connected directed graph
- Score each edge independently
- Find maximal spanning tree

```
this

is example

this

is example

this

is example

5

an

an

an
```
Graph-based vs. Transition Based

- **Transition-based**
  - + Easily condition on infinite tree context (structured prediction)
  - - Greedy search algorithm causes short-term mistakes

- **Graph-based**
  - + Can find exact best global solution via DP algorithm
  - - Have to make local independence assumptions
Chu-Liu-Edmonds
(Chu and Liu 1965, Edmonds 1967)

• We have a graph and want to find its spanning tree

• **Greedily select** the best incoming edge to each node (and subtract its score from all incoming edges)

• If there are cycles, select a cycle and **contract** it into a single node

• **Recursively call** the algorithm on the graph with the contracted node

• **Expand** the contracted node, deleting an edge appropriately
Chu-Liu-Edmonds (1): Find the Best Incoming

(Figure Credit: Jurafsky and Martin)
Chu-Liu-Edmonds (2): Subtract the Max for Each

(Figure Credit: Jurafsky and Martin)
Chu-Liu-Edmonds (3): Contract a Node

(Figure Credit: Jurafsky and Martin)
Chu-Liu-Edmonds (4): Recursively Call Algorithm
Chu-Liu-Edmonds (5): Expand Nodes and Delete Edge

(Figure Credit: Jurafsky and Martin)
Other Dynamic Programs

- **Eisner’s Algorithm** (Eisner 1996):
  - A dynamic programming algorithm to combine together trees in $O(n^3)$
  - Creates *projective* dependency trees (Chu-Liu-Edmonds is *non-projective*)

- **Tarjan’s Algorithm** (Tarjan 1979, Gabow and Tarjan 1983):
  - Like Chu-Liu-Edmonds, but better asymptotic runtime $O(m + n \log n)$
Training Algorithm
(McDonald et al. 2005)

• Basically use **structured hinge loss** (covered in structured prediction class)

• Find the highest scoring tree, penalizing each correct edge by the margin

• If the found tree is not equal to the correct tree, update parameters using hinge loss
Features for Graph-based Parsing (McDonald et al. 2005)

- What features did we use before neural nets?

**a) Basic Uni-gram Features**
- p-word, p-pos
- p-word
- p-pos
- c-word, c-pos
- c-word
- c-pos

**b) Basic Big-ram Features**
- p-word, p-pos, c-word, c-pos
- p-pos, c-word, c-pos
- p-word, c-word, c-pos
- p-word, p-pos, c-pos
- p-pos, c-word, word
- p-pos, c-word
- p-pos, c-pos

**c) In Between POS Features**
- p-pos, b-pos, c-pos
- Surrounding Word POS Features
- p-pos, p-pos+1, c-pos-1, c-pos
- p-pos-1, p-pos, c-pos-1, c-pos
- p-pos, p-pos+1, c-pos, c-pos+1
- p-pos-1, p-pos, c-pos, c-pos+1

Table 1: Features used by system. p-word: word of parent node in dependency tree. c-word: word of child node. p-pos: POS of parent node. c-pos: POS of child node. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child nodes.

- All conjoined with arc direction and arc distance
- Also use POS combination features
- Also represent words w/ prefix if they are long
Higher-order Dependency Parsing
(e.g. Zhang and McDonald 2012)

- Consider multiple edges at a time when calculating scores

<table>
<thead>
<tr>
<th>First Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>I saw a girl with a telescope</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Second Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>I saw a girl with a telescope</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Third Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>I saw a girl with a telescope</td>
</tr>
</tbody>
</table>

• + Can extract more expressive features
• - Higher computational complexity, approximate search necessary
Neural Models for Graph-based Parsing
Neural Feature Combinators
(Pei et al. 2015)

- Extract traditional features, let NN do feature combination
  - Similar to Chen and Manning (2014)’s transition-based model
- Use cube + tanh activation function
- Use averaged embeddings of phrases
- Use second-order features
Phrase Embeddings
(Pei et al. 2015)

• Motivation: words surrounding or between head and dependent are important clues

• Take average of embeddings
Do Neural Feature Combinators Help?  
(Pei et al. 2015)

• Yes!
  
  • 1st-order: LAS 90.39->91.37, speed 26 sent/sec
  • 2nd-order: LAS 91.06->92.13, speed 10 sent/sec
  • 2nd-order neural better than 3rd-order non-neural at UAS
BiLSTM Feature Extractors
(Kipperwasser and Goldberg 2016)

- Simpler and better accuracy than manual extraction
BiAffine Classifier
(Dozat and Manning 2017)

\[
\begin{align*}
    h_i^{(arc-dep)} &= \text{MLP}^{(arc-dep)}(r_i) \\
    h_j^{(arc-head)} &= \text{MLP}^{(arc-head)}(r_j) \\
    s_i^{(arc)} &= H^{(arc-head)} U^{(1)} h_i^{(arc-dep)} \\
    &\quad + H^{(arc-head)} u^{(2)}
\end{align*}
\]

Learn specific representations for head/dependent for each word

Calculate score of each arc

- Just optimize the likelihood of the parent, no structured training
  - This is a local model, with global decoding using MST at the end
- Best results (with careful parameter tuning) on universal dependencies parsing task
Global Training

- Previously: margin-based global training, local probabilistic training
- What about global probabilistic models?

\[
P(Y \mid X) = \frac{e^{\sum_{j=1}^{|Y|} S(y_j \mid X,y_1,\ldots,y_{j-1})}}{\sum_{\tilde{Y} \in V^*} e^{\sum_{j=1}^{|\tilde{Y}|} S(\tilde{y}_j \mid X,\tilde{y}_1,\ldots,\tilde{y}_{j-1})}}
\]

- Algorithms for calculating partition functions:
  - **Projective parsing**: Eisner algorithm is a bottom-up CKY-style algorithm for dependencies (Eisner et al. 1996)
  - **Non-projective parsing**: Matrix-tree theorem can compute marginals over directed graphs (Koo et al. 2007)
- Applied to neural models in Ma et al. (2017)
Dynamic Programming for Phrase Structure Parsing
Phrase Structure Parsing

• Models to calculate phrase structure

[Diagram showing the sentence structure with labels: S, VP, NP, PP, PRP, VBD, DT, NN, IN, DT, NN]

• Important insight: parsing is similar to tagging
  • Tagging is search in a graph for the best path
  • Parsing is search in a hyper-graph for the best tree

The sentence: I saw a girl with a telescope
What is a Hyper-Graph?

- The “degree” of an edge is the number of children

<table>
<thead>
<tr>
<th>Degree 1</th>
<th>Degree 2</th>
<th>Degree 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP 0,1</td>
<td>VP 1,7</td>
<td>VP 1,7</td>
</tr>
<tr>
<td>VBD 1,2</td>
<td>VBD 1,2</td>
<td>VBD 1,2</td>
</tr>
<tr>
<td><strong>saw</strong></td>
<td>NP 2,7</td>
<td>NP 2,4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PP 4,7</td>
</tr>
</tbody>
</table>

- The degree of a hypergraph is the maximum degree of its edges

- A graph is a hypergraph of degree 1!
Tree Candidates as Hypergraphs

- With edges in one tree or another

Two choices! Choose red, get the first tree. Choose blue, get the second tree.

I saw a girl with a telescope.
Weighted Hypergraphs

• Like graphs, can add weights to hypergraph edges

• Generally negative log probability of production

\[-\log(P(S \rightarrow NP \ VP))\]
\[-\log(P(VP \rightarrow VBD \ NP \ PP))\]
\[-\log(P(VP \rightarrow VBD \ NP))\]
\[
\log(P(PR \rightarrow "I"))
\]

I saw a girl with a telescope
Hypergraph Search: CKY Algorithm

- Find the highest-scoring tree given a CFG grammar
- Create a hypergraph containing all candidates for a binarized grammar, do hypergraph search

Analogous to Viterbi algorithm, but Viterbi is over graphs, CKY is over hyper-graphs

```
score(e_1) = 
-log(P(VP \rightarrow VBD NP PP)) +
best_score[VP1,2] +
best_score[NP2,4] +
best_score[NP2,7]

score(e_2) =
-log(P(VP \rightarrow VBD NP)) +
best_score[VP1,2] +
best_score[VP2,7]

best_edge[VP1,7] = \arg\min_{e_1,e_2} \text{score}
best_score[VP1,7] = 
\text{score}(\text{best_edge}[VP1,7])
```
Hypergraph Partition Function: Inside-outside Algorithm

- Find the marginal probability of each span given a CFG grammar
- Partition function us probability of the top span
- Same as CKY, except we logsumexp instead of max
- Analogous to forward-backward algorithm, but forward-backward is over graphs, inside-outside is over hyper-graphs
Neural CRF Parsing
(Durrett and Klein 2015)

- Predict score of each span using FFNN
- Do discrete structured inference using CKY, inside-outside
Span Labeling
(Stern et al. 2017)

• Simple idea: try to decide whether span is constituent in tree or not

(a) Execution of the top-down parsing algorithm.
(b) Output parse tree.

• Allows for various loss functions (local vs. structured), inference algorithms (CKY, top down)
An Alternative: Parse Reranking
An Alternative: Parse Reranking

- You have a nice model, but it’s hard to implement a dynamic programming decoding algorithm
- Try reranking!
  - Generate with an easy-to-decode model
  - Rescore with your proposed model
Examples of Reranking

• Inside-outside recursive neural networks (Le and Zuidema 2014)

• Parsing as language modeling (Choe and Charniak 2016)

• Recurrent neural network grammars (Dyer et al. 2016)
A Word of Caution about Reranking! (Fried et al. 2017)

- Your reranking model got SOTA results, great!
- But, it might be an effect of model combination (which we know works very well)
  - The model generating the parses **prunes down the search space**
  - The reranking model chooses the best parse **only in that space**!

<table>
<thead>
<tr>
<th>Candidates</th>
<th>Scoring models</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RD</td>
<td>RG</td>
<td>RD + RG</td>
<td></td>
</tr>
<tr>
<td>RD</td>
<td>92.22</td>
<td>93.45</td>
<td>93.87</td>
<td></td>
</tr>
<tr>
<td>RG</td>
<td>90.24</td>
<td>89.55</td>
<td>90.53</td>
<td></td>
</tr>
<tr>
<td>RD U RG</td>
<td>92.22</td>
<td>92.78</td>
<td>93.92</td>
<td></td>
</tr>
</tbody>
</table>
Questions?