Overview

- Recap: global linear models
- Dependency parsing
- GLMs for dependency parsing
- Eisner’s parsing algorithm
- Results from McDonald (2005)

Putting it all Together

- $\mathcal{X}$ is set of sentences, $\mathcal{Y}$ is set of possible outputs (e.g. trees)
- Need to learn a function $F : \mathcal{X} \to \mathcal{Y}$
- $\text{GEN}, f, w$ define

\[
F(x) = \arg \max_{y \in \text{GEN}(x)} f(x, y) \cdot w
\]

Choose the highest scoring candidate as the most plausible structure

- Given examples $(x_i, y_i)$, how to set $w$?
She announced a program to promote safety in trucks and vans

\[
\begin{array}{ccccccc}
\downarrow f & \downarrow f & \downarrow f & \downarrow f & \downarrow f & \downarrow f & \downarrow f \\
(1, 1, 3, 5) & (2, 0, 0, 5) & (1, 0, 1, 5) & (0, 0, 3, 0) & (0, 1, 0, 5) & (0, 0, 1, 5) \\
\end{array}
\]

\[
\downarrow f \cdot w & \downarrow f \cdot w & \downarrow f \cdot w & \downarrow f \cdot w & \downarrow f \cdot w & \downarrow f \cdot w \\
13.6 & 12.2 & 12.1 & 3.3 & 9.4 & 11.1 \\
\]

\[
\text{arg max} 
\]

5

A Variant of the Perceptron Algorithm

Inputs: Training set \((x_i, y_i)\) for \(i = 1 \ldots n\)

Initialization: \(w = 0\)

Define: \(F(x) = \arg\max_{y \in \text{GEN}(x)} f(x, y) \cdot w\)

Algorithm: For \(t = 1 \ldots T, i = 1 \ldots n\)

\[z_i = F(x_i)\]

If \((z_i \neq y_i)\)

\[w = w + f(x_i, y_i) - f(x_i, z_i)\]

Output: Parameters \(w\)

A tagged sentence with \(n\) words has \(n\) history/tag pairs

Hispaniola/NNP quickly/RB became/VB an_DT important/JJ base/NN

<table>
<thead>
<tr>
<th>History</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>* *</td>
<td>{Hispaniola, quickly, \ldots}</td>
</tr>
<tr>
<td>* NNP</td>
<td>{Hispaniola, quickly, \ldots}</td>
</tr>
<tr>
<td>NNP RB</td>
<td>{Hispaniola, quickly, \ldots}</td>
</tr>
<tr>
<td>RB VB</td>
<td>{Hispaniola, quickly, \ldots}</td>
</tr>
<tr>
<td>VP DT</td>
<td>{Hispaniola, quickly, \ldots}</td>
</tr>
<tr>
<td>DT JJ</td>
<td>{Hispaniola, quickly, \ldots}</td>
</tr>
</tbody>
</table>

Define global features through local features:

\[f(t_{[1:n]}, w_{[1:n]}) = \sum_{i=1}^{n} g(h_i, t_i)\]

where \(t_i\) is the \(i\)'th tag, \(h_i\) is the \(i\)'th history

Global and Local Features

- Typically, local features are indicator functions, e.g.,

\[g_{101}(h, t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \text{\text{ing}} \text{ and } t = \text{VBG} \\ 0 & \text{otherwise} \end{cases}\]

- and global features are then counts,

\[f_{101}(w_{[1:n]}, t_{[1:n]}) = \text{Number of times a word ending in } \text{\text{ing}} \text{ is tagged as } \text{VBG} \text{ in } (w_{[1:n]}, t_{[1:n]})\]
Putting it all Together

- \( \text{GEN}(w_{1:n}) \) is the set of all tagged sequences of length \( n \)
- \( \text{GEN}, f, w \) define
  \[
  F(w_{1:n}) = \arg \max_{t_{1:n} \in \text{GEN}(w_{1:n})} w \cdot f(w_{1:n}, t_{1:n})
  \]
  \[
  = \arg \max_{t_{1:n} \in \text{GEN}(w_{1:n})} w \cdot \sum_{i=1}^{n} g(h_i, t_i)
  \]
  \[
  = \arg \max_{t_{1:n} \in \text{GEN}(w_{1:n})} \sum_{i=1}^{n} w \cdot g(h_i, t_i)
  \]
- Some notes:
  - Score for a tagged sequence is a sum of local scores
  - Dynamic programming can be used to find the \( \arg \max \)!
    (because history only considers the previous two tags)

Training a Tagger Using the Perceptron Algorithm

**Inputs:** Training set \( (w_{1:n}^i, t_{1:n}^i) \) for \( i = 1 \ldots n \).

**Initialization:** \( w = 0 \)

**Algorithm:** For \( t = 1 \ldots T, i = 1 \ldots n \)

\[
 z_{1:n} = \arg \max_{u_{1:n} \in T_{1:n}} w \cdot f(w_{1:n}^i, u_{1:n})
 \]

\( z_{1:n} \) can be computed with the dynamic programming (Viterbi) algorithm

If \( z_{1:n} \neq t_{1:n}^i \) then

\[
 w = w + f(w_{1:n}^i, t_{1:n}^i) - f(w_{1:n}^i, z_{1:n}^i)
 \]

**Output:** Parameter vector \( w \).

A Variant of the Perceptron Algorithm

**Inputs:** Training set \( (x_i, y_i) \) for \( i = 1 \ldots n \)

**Initialization:** \( w = 0 \)

**Define:**

\[
 F(x) = \arg \max_{y \in \text{GEN}(x)} f(x, y) \cdot w
 \]

**Algorithm:** For \( t = 1 \ldots T, i = 1 \ldots n \)

\[
 z_i = F(x_i)
 \]

If \( z_i \neq y_i \) \( w = w + f(x_i, y_i) - f(x_i, z_i) \)

**Output:** Parameters \( w \)

Overview

- Recap: global linear models
- Dependency parsing
- GLMs for dependency parsing
- Eisner’s parsing algorithm
- Results from McDonald (2005)
Unlabeled Dependency Parses

- root is a special root symbol
- Each dependency is a pair \((h, m)\) where \(h\) is the index of a head word, \(m\) is the index of a modifier word. In the figures, we represent a dependency \((h, m)\) by a directed edge from \(h\) to \(m\).
- Dependencies in the above example are \((0, 2)\), \((2, 1)\), \((2, 4)\), and \((4, 3)\). (We take 0 to be the root symbol.)

All Dependency Parses for John saw Mary

A More Complex Example

- The dependency arcs form a directed tree, with the root symbol at the root of the tree.
  (Definition: A directed tree rooted at root is a tree, where for every word \(w\) other than the root, there is a directed path from root to \(w\).)
- There are no “crossing dependencies”.
  Dependency structures with no crossing dependencies are sometimes referred to as projective structures.

Conditions on Dependency Structures
Labeled Dependency Parses

- Similar to unlabeled structures, but each dependency is a triple \((h, m, l)\) where \(h\) is the index of a head word, \(m\) is the index of a modifier word, and \(l\) is a label. In the figures, we represent a dependency \((h, m, l)\) by a directed edge from \(h\) to \(m\) with a label \(l\).

- For most of this lecture we’ll stick to unlabeled dependency structures.

Extracting Dependency Parses from Treebanks

- There’s recently been a lot of interest in dependency parsing. For example, the CoNLL 2006 conference had a “shared task” where 12 languages were involved (Arabic, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish, Turkish). 19 different groups developed dependency parsing systems. CoNLL 2007 had a similar shared task. Google for “conll 2006 shared task” for more details. For a recent PhD thesis on the topic, see Ryan McDonald, Discriminative Training and Spanning Tree Algorithms for Dependency Parsing, University of Pennsylvania.

- For some languages, e.g., Czech, there are “dependency banks” available which contain training data in the form of sentences paired with dependency structures

- For other languages, we have treebanks from which we can extract dependency structures, using lexicalized grammars described earlier in the course (see Parsing and Syntax 2)
Efficiency of Dependency Parsing

- PCFG parsing is $O(n^3G^3)$ where $n$ is the length of the sentence, $G$ is the number of non-terminals in the grammar.
- Lexicalized PCFG parsing is $O(n^5G^3)$ where $n$ is the length of the sentence, $G$ is the number of non-terminals in the grammar. (With the algorithms we’ve seen—it is possible to do a little better than this.)
- Unlabeled dependency parsing is $O(n^3)$. (See part 4 of these slides for the algorithm.)

Overview

- Recap: global linear models
- Dependency parsing
- Global Linear Models (GLMs) for dependency parsing
- Eisner’s parsing algorithm
- Results from McDonald (2005)

GLMs for Dependency parsing

- $x$ is a sentence
- $\text{GEN}(x)$ is set of all dependency structures for $x$
- $f(x, y)$ is a feature vector for a sentence $x$ paired with a dependency parse $y$

GLMs for Dependency parsing

- To run the perceptron algorithm, we must be able to efficiently calculate
  $$\arg\max_{y \in \text{GEN}(x)} w \cdot f(x, y)$$
- Local feature vectors: define
  $$f(x, y) = \sum_{(h, m) \in y} g(x, h, m)$$
  where $g(x, h, m)$ maps a sentence $x$ and a dependency $(h, m)$ to a local feature vector
- Can then efficiently calculate
  $$\arg\max_{y \in \text{GEN}(x)} w \cdot f(x, y) = \arg\max_{y \in \text{GEN}(x)} \sum_{(h, m) \in y} w \cdot g(x, h, m)$$
Definition of Local Feature Vectors

- \( g(x, h, m) \) maps a sentence \( x \) and a dependency \((h, m)\) to a local feature vector

- Features from McDonald et al. (2005):
  - Note: define \( w_i \) to be the \( i \)'th word in the sentence, \( t_i \) to be the part-of-speech (POS) tag for the \( i \)'th word.
  - **Unigram features:** Identity of \( w_h \). Identity of \( w_m \). Identity of \( t_h \). Identity of \( t_m \).
  - **Bigram features:** Identity of the 4-tuple \( \langle w_h, w_m, t_h, t_m \rangle \). Identity of sub-sets of this 4-tuple, e.g., identity of the pair \( \langle w_h, w_m \rangle \).
  - **Contextual features:** Identity of the 4-tuple \( \langle t_h, t_{h+1}, t_{m-1}, t_m \rangle \). Similar features which consider \( t_{h-1} \) and \( t_{m+1} \), giving 4 possible feature types.
  - **In-between features:** Identity of triples \( \langle t_h, t, t_m \rangle \) for any tag \( t \) seen between words \( h \) and \( m \).

Eisner’s Algorithm for Dependency Parsing

- Runs in \( O(n^3) \) time for a sentence of length \( n \)
- Algorithm is similar to the dynamic programming algorithm for PCFGs, but represents constituents in a novel way
- The problem: find
  \[
  \arg\max_{y \in \text{GEN}(x)} \sum_{(h, m) \in y} S(h, m)
  \]
  where \( x \) is a sentence, \( \text{GEN}(x) \) is the set of all dependency trees for \( x \), and \( S(h, m) \) is the score of dependency \((h, m)\). In our case,
  \[
  S(h, m) = w \cdot g(x, h, m)
  \]

Overview

- Recap: global linear models
- Dependency parsing
- Global Linear Models (GLMs) for dependency parsing
- Eisner’s parsing algorithm
- Results from McDonald (2005)

Complete Constituents

- A complete constituent with direction \( \rightarrow \) for words \( w_s \ldots w_t \) is a set of dependencies \( D \) such that:
  - Every word in \( w_{s+1} \ldots w_t \) is a modifier to some word in \( w_s \ldots w_t \).
  - The dependencies in \( D \) form a well formed dependency sub-parse: i.e., there are no crossing dependencies, or cycles. No dependencies in \( D \) involve words other than \( w_s \ldots w_t \).
  - \( w_s \) is the head of at least one dependency.
- Note: this means that the dependencies in \( D \) form a directed tree that spans all words \( w_s \ldots w_t \), with \( w_s \) at the root of the tree.
**Complete Constituents**

- A *complete constituent* with direction $\leftarrow$ for words $w_s \ldots w_t$ is a set of dependencies $D$ such that:
  - Every word in $w_s \ldots w_{t-1}$ is a modifier to some word in $w_s \ldots w_t$.
  - The dependencies in $D$ form a well formed dependency sub-parse: i.e., there are no crossing dependencies, or cycles. No dependencies in $D$ involve words other than $w_s \ldots w_t$.
  - $w_t$ is the head of at least one dependency.

- Note: this means that the dependencies in $D$ form a directed tree that spans all words $w_s \ldots w_t$, with $w_t$ at the root of the tree.

**Incomplete Constituents**

- An *incomplete constituent* with direction $\leftarrow$ for words $w_s \ldots w_t$ is a set of dependencies $D$ such that:
  - Every word in $w_s \ldots w_{t-1}$ is a modifier to some word in $w_s \ldots w_t$.
  - The dependencies in $D$ form a well formed dependency sub-parse: i.e., there are no crossing dependencies, or cycles. No dependencies in $D$ involve words other than $w_s \ldots w_t$.
  - $w_t$ is the head of at least one dependency.
  - A new condition: there is a dependency $(t,s)$ in $D$.

- Note: any incomplete constituent is also a complete constituent

---

**The Dynamic Programming Table**

- $C[s][t][d][c]$ is the highest score for any constituent that:
  - Spans words $w_s \ldots w_t$
  - Has direction $d$ (either $\rightarrow$ or $\leftarrow$)
  - Has type $c$ ($c = 0$ for incomplete constituents, $c = 1$ for complete constituents)

- Base case for the dynamic programming algorithm:
  
  for $s = 1 \ldots n$, $C[s][s][\rightarrow][1] = C[s][s][\leftarrow][1] = 0.0$
Intuition: Creating Incomplete Constituents

- We can form an incomplete constituent spanning words $w_s \ldots w_t$ by combining two complete constituents.

Creating Incomplete Constituents

- First case: for any $s, t$ such that $1 \leq s < t \leq n$,
  \[ C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r + 1][t][\leftarrow][1] + S(t, s)) \]

  Intuition: combine two complete constituents to form an incomplete constituent

- Second case: for any $s, t$ such that $1 \leq s < t \leq n$,
  \[ C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r + 1][t][\leftarrow][1] + S(s, t)) \]

Intuition: Creating Complete Constituents

- We can form a complete constituent spanning words $w_s \ldots w_t$ by combining an incomplete and a complete constituent.

Creating Complete Constituents

- First case: for any $s, t$ such that $1 \leq s < t \leq n$,
  \[ C[s][t][\leftarrow][1] = \max_{s \leq r < t} (C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0]) \]

  Intuition: combine one complete constituent, one incomplete constituent, to form a complete constituent

- Second case: for any $s, t$ such that $1 \leq s < t \leq n$,
  \[ C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1]) \]
The Full Algorithm

Initialization:

\[
\text{for } s = 0 \ldots n, \quad C[s][s][\rightarrow][1] = C[s][s][\leftarrow][1] = 0.0
\]

\[
\text{for } k = 1 \ldots n + 1
\]

\[
\text{for } s = 0 \ldots n
\]

\[
t = s + k
\]

\[
\text{if } t > n \text{ then break}
\]

% First: create incomplete items

\[
C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r + 1][t][\leftarrow][1] + S(t, s))
\]

\[
C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r + 1][t][\leftarrow][1] + S(s, t))
\]

% Second: create incomplete items

\[
C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][1] + C[r][t][\leftarrow][0])
\]

\[
C[s][t][\leftarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\leftarrow][1])
\]

Return \(C[0][n][\rightarrow][1]\) as the highest score for any parse

Overview

- Recap: global linear models
- Dependency parsing
- Global Linear Models (GLMs) for dependency parsing
- Eisner’s parsing algorithm
- Results from McDonald (2005)

Results from McDonald (2005)

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins (1997)</td>
<td>91.4%</td>
</tr>
<tr>
<td>1st order dependency</td>
<td>90.7%</td>
</tr>
<tr>
<td>2nd order dependency</td>
<td>91.5%</td>
</tr>
</tbody>
</table>

- Accuracy is percentage of correct unlabeled dependencies
- Collins (1997) is result from a lexicalized context-free parser, with dependencies extracted from the parser’s output
- 1st order dependency is the method just described. 2nd order dependency is a model that uses richer representations.
- Advantages of the dependency parsing approaches: simplicity, efficiency (\(O(n^3)\) parsing time).

Extensions

- 2nd-order dependency parsing
- Non-projective dependency structures