6.864 (Fall 2007)

Global Linear Models: Part III

Overview

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

Three Components of Global Linear Models

- $\mathbf{f}$ is a function that maps a structure (x, y) to a **feature vector** $\mathbf{f}(x, y) \in \mathbb{R}^d$
- $\mathbf{GEN}$ is a function that maps an input x to a set of **candidates** $\mathbf{GEN}(x)$
- $\mathbf{w}$ is a parameter vector (also a member of $\mathbb{R}^d$)
- Training data is used to set the value of $\mathbf{w}$

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 $F : \mathcal{X} \to \mathcal{Y}$
- $\mathbf{GEN}$, $\mathbf{f}$, $\mathbf{w}$ define

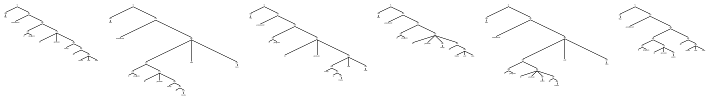
$$F(x) = \arg\max_{y \in \mathbf{GEN}(x)} \mathbf{f}(x, y) \cdot \mathbf{w}$$

Choose the highest scoring candidate as the most plausible structure

- Given examples (x_i, y_i), how to set $\mathbf{w}$?

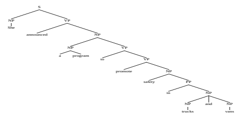
She announced a program to promote safety in trucks and vans

$\Downarrow$ **GEN**



$\Downarrow \mathbf{f}$	$\Downarrow \mathbf{f}$	$\Downarrow \mathbf{f}$	$\Downarrow \mathbf{f}$	$\Downarrow \mathbf{f}$	$\Downarrow \mathbf{f}$
$\langle 1,1,3,5\rangle$	$\langle 2,0,0,5\rangle$	$\langle 1,0,1,5\rangle$	$\langle 0,0,3,0\rangle$	$\langle 0,1,0,5\rangle$	$\langle 0,0,1,5\rangle$
$\Downarrow \mathbf{f}\cdot\mathbf{w}$	$\Downarrow \mathbf{f}\cdot\mathbf{w}$	$\Downarrow \mathbf{f}\cdot\mathbf{w}$	$\Downarrow \mathbf{f}\cdot\mathbf{w}$	$\Downarrow \mathbf{f}\cdot\mathbf{w}$	$\Downarrow \mathbf{f}\cdot\mathbf{w}$
13.6	12.2	12.1	3.3	9.4	11.1

$\Downarrow \arg\max$



A Variant of the Perceptron Algorithm

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

Initialization: $\mathbf{w} = 0$

Define: $F(x) = \text{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{f}(x, y) \cdot \mathbf{w}$

Algorithm: For $t = 1 \dots T$, $i = 1 \dots n$
$z_i = F(x_i)$
If $(z_i \neq y_i)$ $\quad \mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y_i) - \mathbf{f}(x_i, z_i)$

Output: Parameters $\mathbf{w}$

A tagged sentence with n words has n history/tag pairs

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/NN

History				Tag
t_{-2}	t_{-1}	$w_{[1:n]}$	i	t
*	*	$\langle Hispaniola, quickly, \dots, \rangle$	1	NNP
*	NNP	$\langle Hispaniola, quickly, \dots, \rangle$	2	RB
NNP	RB	$\langle Hispaniola, quickly, \dots, \rangle$	3	VB
RB	VB	$\langle Hispaniola, quickly, \dots, \rangle$	4	DT
VP	DT	$\langle Hispaniola, quickly, \dots, \rangle$	5	JJ
DT	JJ	$\langle Hispaniola, quickly, \dots, \rangle$	6	NN

Define global features through local features:

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

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.,

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

- and global features are then counts,

$f_{101}(w_{[1:n]}, t_{[1:n]})$ = Number of times a word ending in `ing` is tagged as `VBG` in $(w_{[1:n]}, t_{[1:n]})$

Putting it all Together

- $\mathbf{GEN}(w_{[1:n]})$ is the set of all tagged sequences of length n
- $\mathbf{GEN}$, $\mathbf{f}$, $\mathbf{w}$ define

$$
\begin{aligned}
F(w_{[1:n]}) &= \arg\max_{t_{[1:n]} \in \mathbf{GEN}(w_{[1:n]})} \mathbf{w} \cdot \mathbf{f}(w_{[1:n]}, t_{[1:n]}) \\
&= \arg\max_{t_{[1:n]} \in \mathbf{GEN}(w_{[1:n]})} \mathbf{w} \cdot \sum_{i=1}^{n} \mathbf{g}(h_i, t_i) \\
&= \arg\max_{t_{[1:n]} \in \mathbf{GEN}(w_{[1:n]})} \sum_{i=1}^{n} \mathbf{w} \cdot \mathbf{g}(h_i, t_i)
\end{aligned}
$$

- Some notes:
  - Score for a tagged sequence is a sum of local scores
  - **Dynamic programming can be used to find the** argmax**!** (because history only considers the previous two tags)

A Variant of the Perceptron Algorithm

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

Initialization: $\mathbf{w} = 0$

Define: $F(x) = \text{argmax}_{y \in \mathbf{GEN}(x)} \mathbf{f}(x, y) \cdot \mathbf{w}$

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

$z_i = F(x_i)$

If $(z_i \neq y_i)$ $\quad \mathbf{w} = \mathbf{w} + \mathbf{f}(x_i, y_i) - \mathbf{f}(x_i, z_i)$

Output: Parameters $\mathbf{w}$

Training a Tagger Using the Perceptron Algorithm

Inputs: Training set $(w^i_{[1:n_i]}, t^i_{[1:n_i]})$ for $i = 1 \ldots n$.

Initialization: $\mathbf{w} = 0$

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

$$z_{[1:n_i]} = \arg\max_{u_{[1:n_i]} \in \mathcal{T}^{n_i}} \mathbf{w} \cdot \mathbf{f}(w^i_{[1:n_i]}, u_{[1:n_i]})$$

$z_{[1:n_i]}$ can be computed with the dynamic programming (Viterbi) algorithm

If $z_{[1:n_i]} \neq t^i_{[1:n_i]}$ then

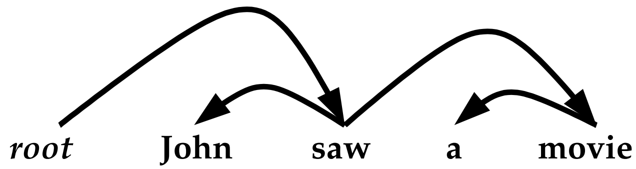
$$\mathbf{w} = \mathbf{w} \quad + \quad \mathbf{f}(w^i_{[1:n_i]}, t^i_{[1:n_i]}) - \mathbf{f}(w^i_{[1:n_i]}, z_{[1:n_i]})$$

Output: Parameter vector $\mathbf{w}$.

Overview

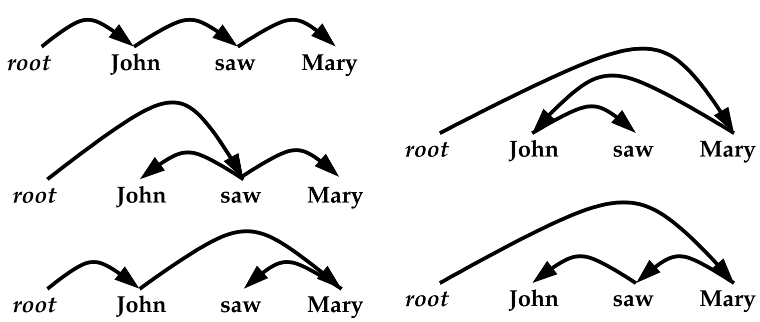
- Recap: global linear models
- Dependency parsing
- GLMs for dependency parsing
- Eisner's parsing algorithm
- Results from McDonald (2005)

Unlabeled Dependency Parses

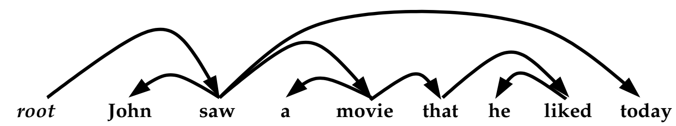


- `root` is a special *root* symbol
- Each dependency is a pair (h, m) where h is the index of a head word, m is the index of a modifier word. In the figures, we represent a dependency (h, m) by a directed edge from h to m.
- Dependencies in the above example are $(0, 2)$, $(2, 1)$, $(2, 4)$, and $(4,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

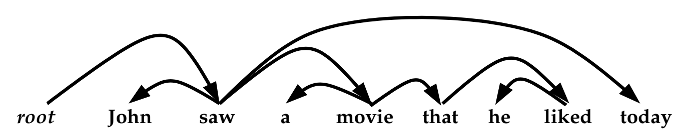
All Dependency Parses for *John saw Mary*



A More Complex Example



Conditions on Dependency Structures



- 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”.

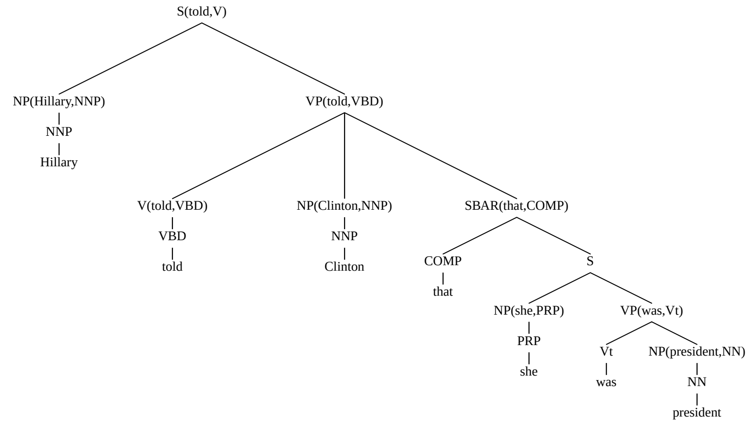
  Dependency structures with no crossing dependencies are sometimes referred to as **projective** structures.

Labeled Dependency Parses

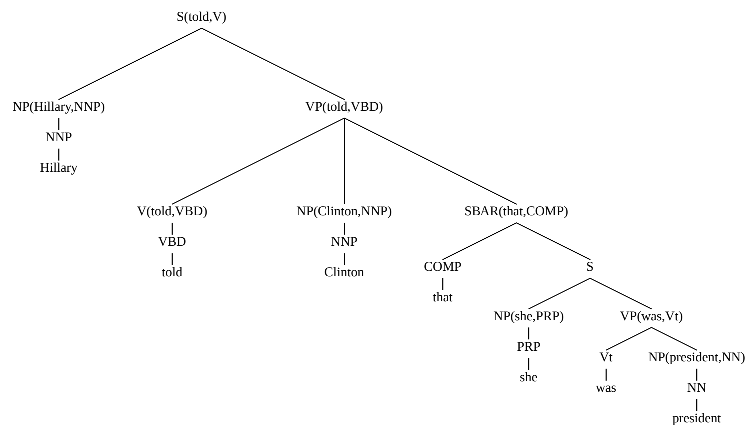
- Similar to unlabeled structures, but each dependency is a triple (h, m, l) where h is the index of a head word, m is the index of a modifier word, and l is a label. In the figures, we represent a dependency (h, m, l) by a directed edge from h to m with a label l.
- For most of this lecture we'll stick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



(__	__	told	VBD	TOP	S	__	SPECIAL)
(told	VBD	Hillary	NNP	S	VP	NP	LEFT)
(told	VBD	Clinton	NNP	VP	VBD	NP	RIGHT)
(told	VBD	that	COMP	VP	VBD	SBAR	RIGHT)
(that	COMP	was	Vt	SBAR	COMP	S	RIGHT)
(was	Vt	she	PRP	S	VP	NP	LEFT)
(was	Vt	president	NP	VP	Vt	NP	RIGHT)



Unlabeled Dependencies:

(0,2) (for `root` $\rightarrow$ told)
(2,1) (for told $\rightarrow$ Hillary)
(2,3) (for told $\rightarrow$ Clinton)
(2,4) (for told $\rightarrow$ that)
(4,6) (for that $\rightarrow$ was)
(6,5) (for was $\rightarrow$ she)
(6,7) (for was $\rightarrow$ president)

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
- $\mathbf{GEN}(x)$ is set of all dependency structures for x
- $\mathbf{f}(x, y)$ 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

$$\arg \max_{y \in \mathbf{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y)$$

- Local feature vectors: define

$$\mathbf{f}(x, y) = \sum_{(h,m) \in y} \mathbf{g}(x, h, m)$$

  where $\mathbf{g}(x, h, m)$ maps a sentence x and a dependency (h, m) to a local feature vector

- Can then efficiently calculate

$$\arg \max_{y \in \mathbf{GEN}(x)} \mathbf{w} \cdot \mathbf{f}(x, y) = \arg \max_{y \in \mathbf{GEN}(x)} \sum_{(h,m) \in y} \mathbf{w} \cdot \mathbf{g}(x, h, m)$$

Definition of Local Feature Vectors

- $\mathbf{g}(x, h, m)$ 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.

Overview

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

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

$$\arg \max_{y \in \mathbf{GEN}(x)} \sum_{(h,m) \in y} \mathbf{S}(h, m)$$

where x is a sentence, $\mathbf{GEN}(x)$ is the set of all dependency trees for x, and $\mathbf{S}(h, m)$ is the score of dependency (h, m). In our case,

$$\mathbf{S}(h, m) = \mathbf{w} \cdot \mathbf{g}(x, h, m)$$

Complete Constituents

- A *complete consituent* 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 consituent* 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 consituent* 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.
  - A new condition: there is a dependency (s, t) in D.
- Note: any incomplete constituent is also a complete constituent

Incomplete Constituents

- An *incomplete consituent* 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:

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

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 $1 \leq s < t \leq n$,

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

Intuition: combine two complete constituents to form an incomplete constituent

- Second case: for any s, t such that $1 \leq s < t \leq n$,

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

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 $1 \leq s < t \leq n$,

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

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

- Second case: for any s, t such that $1 \leq s < t \leq n$,

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

The Full Algorithm

Initialization:

for $s = 0 \ldots n$, $C[s][s][\rightarrow][1] = C[s][s][\leftarrow][1] = 0.0$

for $k = 1 \ldots n+1$
  for $s = 0 \ldots n$
    $t = s + k$
    if $t > n$ then break

    % First: create incomplete items
    $C[s][t][\leftarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] \; + \; C[r+1][t][\leftarrow][1] \; + \mathbf{S}(t,s))$
    $C[s][t][\rightarrow][0] = \max_{s \leq r < t} (C[s][r][\rightarrow][1] \; + \; C[r+1][t][\leftarrow][1] \; + \mathbf{S}(s,t))$

    % Second: create incomplete items
    $C[s][t][\leftarrow][1] = \max_{s \leq r < t} (C[s][r][\leftarrow][1] \; + \; C[r][t][\leftarrow][0])$
    $C[s][t][\rightarrow][1] = \max_{s < r \leq t} (C[s][r][\rightarrow][0] \; + \; C[r][t][\rightarrow][1])$

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)

Method	Accuracy
Collins (1997)	91.4%
1st order dependency	90.7%
2nd order dependency	91.5%

- 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