Lecture 3, COMS E6998-3: Discriminative Dependency Parsing

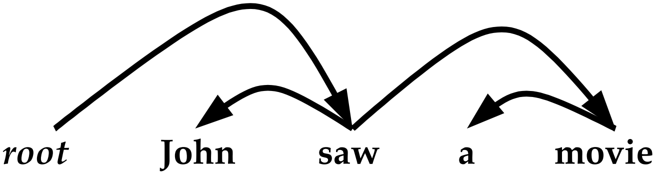
Michael Collins

February 2, 2011

Projects

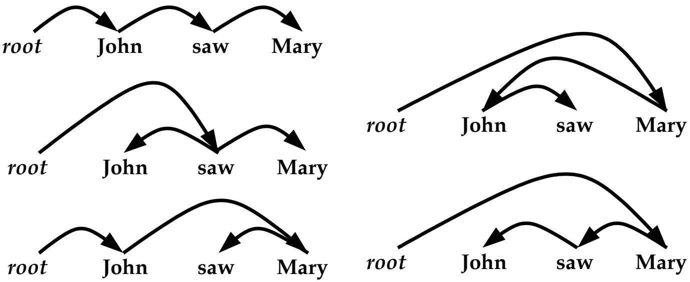
- First deadline: 1 page project proposal by **5pm, Friday February 11th**
- The choice of project is up to you, but it should be clearly related to the course material
- Example projects:
  - Design and implementation of a machine-learning model for some NLP task; the write-up would describe the technical details of the model, as well as experimentation with the model on some dataset
  - Implementation of an approach (or approaches) described in one or more papers in the research literature
  - Possibly also purely "theoretical" projects (no experimentation), although these projects will be less common
- Group projects are allowed (up to a maximum of 3 people)
- We'll expect a 6 page write-up for 1 person projects, 8 pages for 2 person projects, 10 pages for 3 people.

Unlabeled Dependency Parses

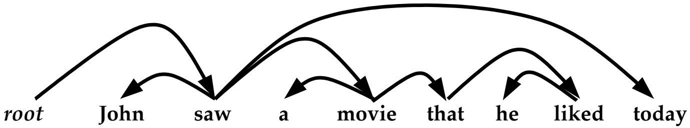


- `root` is a special *root* symbol
- Each dependency is a pair (j, k) where j index of a head word, k is the index of a modifier word. In the figures, we represent a dependency (j, k) by a directed edge from word j to word k
- Dependencies in the above example are $(0, 2)$, $(2, 1)$, $(2, 4)$ and $(4,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

All Dependency Parses for *John saw Mary*



Conditions on Dependency Structures



- The dependency arcs form a *directed tree*, with the `root` symbol at the root of the tree.
- There are no "crossing dependencies".
  Dependency structures with no crossing dependencies are sometimes referred to as **projective** structures.

Notation for Dependency Structures

- Assume $\underline{x}$ is a sequence of words $x_1 \ldots x_m$
- A dependency structure is a vector $\underline{y}$
- First, define the *index set* $\mathcal{I}$ to be the set of all possible dependencies. For example, for $m = 3$,

  $$\mathcal{I} = \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)\}$$

- Then $\underline{y}$ is a vector of values $y(j,k)$ for all $(j,k) \in \mathcal{I}$.
  $y(j,k) = 1$ if the structure contains the dependency (j,k),
  $y(j,k) = 0$ otherwise.
- We use $\mathcal{Y}$ to refer to the set of all possible well-formed vectors $\underline{y}$

Feature Vectors for Dependencies

- $\underline{\phi}(\underline{x}, j, k)$ is a feature vector representing dependency (j, k) for sentence $\underline{x}$
- Example features:
  - Identity of the words x_j and x_k
  - The part-of-speech tags for words x_j and x_k
  - The distance between x_j and x_k
  - Words/tags that surround x_j and x_k
  - etc. etc.

CRFs for Discriminative Dependency Parsing

- We use $\underline{\Phi}(\underline{x}, \underline{y}) \in \mathbb{R}^d$ to refer to a feature vector for an *entire* dependency structure $\underline{y}$
- We then build a log-linear model, very similar to a CRF

$$p(\underline{y}|\underline{x}; \underline{w}) = \frac{\exp\left(\underline{w} \cdot \underline{\Phi}(\underline{x}, \underline{y})\right)}{\sum_{\underline{y}' \in \mathcal{Y}} \exp\left(\underline{w} \cdot \underline{\Phi}(\underline{x}, \underline{y}')\right)}$$

- How do we define $\underline{\Phi}(\underline{x}, \underline{y})$? Answer:

$$\underline{\Phi}(\underline{x}, \underline{y}) = \sum_{(j,k) \in \mathcal{I}} y(j, k) \underline{\phi}(\underline{x}, j, k)$$

where $\underline{\phi}(\underline{x}, j, k)$ is the feature vector for dependency (j, k)

Decoding

- The decoding problem: find

$$
\begin{aligned}
\arg\max_{\underline{y}\in\mathcal{Y}} p(\underline{y}|\underline{x};\underline{w}) &= \arg\max_{\underline{y}\in\mathcal{Y}} \frac{\exp\left(\underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y})\right)}{\sum_{\underline{y}'\in\mathcal{Y}}\exp\left(\underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y}')\right)} \\
&= \arg\max_{\underline{y}\in\mathcal{Y}} \exp\left(\underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y})\right) \\
&= \arg\max_{\underline{y}\in\mathcal{Y}} \underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y}) \\
&= \arg\max_{\underline{y}\in\mathcal{Y}} \underline{w}\cdot\sum_{(j,k)\in\mathcal{I}} y(j,k)\underline{\phi}(\underline{x},j,k) \\
&= \arg\max_{\underline{s}\in\mathcal{Y}} \sum_{(j,k)\in\mathcal{I}} y(j,k)\left(\underline{w}\cdot\underline{\phi}(\underline{x},j,k)\right)
\end{aligned}
$$

- This problem can be solved using dynamic programming, in $O(m^3)$ time, where m is the length of the sentence

Parameter Estimation

- To estimate the parameters, we assume we have a set of n labeled examples, $\{(\underline{x}^i, \underline{y}^i)\}_{i=1}^n$. Each $\underline{x}^i$ is an input sequence $x_1^i \dots x_m^i$, each $\underline{y}^i$ is a dependency structure (i.e., $y^i(j,k) = 1$ if the i'th structure contains a dependency (j,k)).
- We then proceed in exactly the same way as for CRFs
- The *regularized log-likelihood function* is

$$L(\underline{w}) = \sum_{i=1}^{n} \log p(\underline{y}^i | \underline{x}^i; \underline{w}) - \frac{\lambda}{2} ||\underline{w}||^2$$

- The *parameter estimates* are

$$\underline{w}^* = \arg\max_{\underline{w} \in \mathbb{R}^d} \sum_{i=1}^{n} \log p(\underline{y}^i | \underline{x}^i; \underline{w}) - \frac{\lambda}{2} ||\underline{w}||^2$$

Finding the Maximum-Likelihood Estimates

- We'll again use gradient-based optimization methods to find $\underline{w}^*$
- How can we compute the derivatives? As before,

$$\frac{\partial}{\partial w_l} L(\underline{w}) \quad = \quad \sum_i \Phi_l(\underline{x}^i, \underline{y}^i) - \sum_i \sum_{\underline{y} \in \mathcal{Y}} p(\underline{y} | \underline{x}^i; \underline{w}) \Phi_l(\underline{x}^i, \underline{y}) - \lambda w_l$$

- The first term is easily computed, because

$$\sum_i \Phi_l(\underline{x}^i, \underline{y}^i) = \sum_i \sum_{(j,k) \in \mathcal{I}} y^i(j,k) \phi_l(\underline{x}^i, j, k)$$

- The second term involves a sum over $\mathcal{Y}$, and because of this looks nasty...

Calculating Derivatives using Dynamic Programming

- We now consider how to compute the second term:

$$\sum_{\underline{y} \in \mathcal{Y}} p(\underline{y}|\underline{x}^i; \underline{w}) \Phi_l(\underline{x}^i, \underline{y}) = \sum_{\underline{y} \in \mathcal{Y}} p(\underline{y}|\underline{x}^i; \underline{w}) \sum_{(j,k) \in \mathcal{I}} y(j,k) \phi_l(\underline{x}^i, j, k)$$
$$= \sum_{(j,k) \in \mathcal{I}} q^i(j,k) \phi_l(\underline{x}^i, j, k)$$

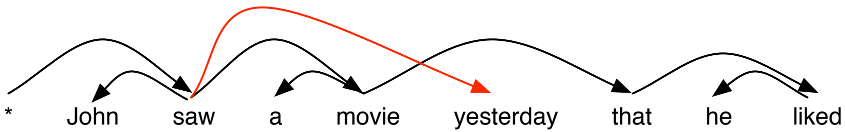
  where

$$q^i(j,k) = \sum_{\underline{y} \in \mathcal{Y} : y(j,k)=1} p(\underline{y}|\underline{x}^i; \underline{w})$$

  (for the full derivation see the notes)

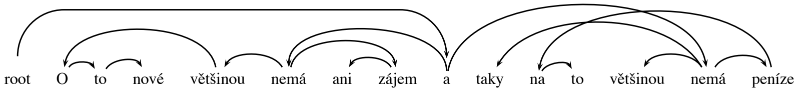
- For a given i, all $q^i(j,k)$ terms can be computed simultaneously in $O(m^3)$ time using dynamic programming.

Non-Projective Dependency Parsing



- We can also consider *non-projective* dependency parses, where crossing dependencies are allowed
- Define $\mathcal{Y}_{np}$ to be the set of all non-projective dependency parses
- Each dependency parse $\underline{y} \in \mathcal{Y}_{np}$ is a vector of values $y(j,k)$ for all $(j,k) \in \mathcal{I}$. $y(j,k) = 1$ if the structure contains the dependency (j,k), $y(j,k) = 0$ otherwise.

An Example from Czech



He is mostly not even interested in the new things and in most cases, he has no money for it either.

(figure taken from McDonald et al, 2005)

CRFs for Non-Projective Structures

- We use $\underline{\Phi}(\underline{x}, \underline{y}) \in \mathbb{R}^d$ to refer to a feature vector for an *entire* dependency structure $\underline{y}$
- We then build a log-linear model, very similar to a CRF

$$p(\underline{y}|\underline{x}; \underline{w}) = \frac{\exp\left(\underline{w} \cdot \underline{\Phi}(\underline{x}, \underline{y})\right)}{\sum_{\underline{y}' \in \mathcal{Y}_{np}} \exp\left(\underline{w} \cdot \underline{\Phi}(\underline{x}, \underline{y}')\right)}$$

- How do we define $\underline{\Phi}(\underline{x}, \underline{y})$? Answer:

$$\underline{\Phi}(\underline{x}, \underline{y}) = \sum_{(j,k) \in \mathcal{I}} y(j,k) \underline{\phi}(\underline{x}, j, k)$$

where $\underline{\phi}(\underline{x}, j, k)$ is the feature vector for dependency (j, k)

Only change from projective parsing: we've replaced the set of projective parses $\mathcal{Y}$, with the set of non-projective parses, $\mathcal{Y}_{np}$

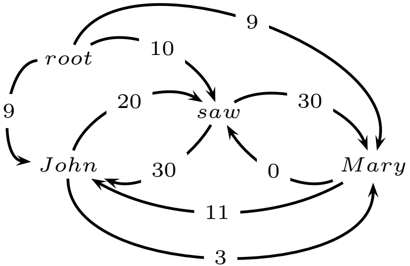
Decoding in Non-Projective Models

- The decoding problem: find

$$\begin{aligned}
\arg\max_{\underline{y}\in\mathcal{Y}_{np}} p(\underline{y}|\underline{x};\underline{w}) &= \arg\max_{\underline{y}\in\mathcal{Y}_{np}} \frac{\exp\left(\underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y})\right)}{\sum_{\underline{y}'\in\mathcal{Y}}\exp\left(\underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y}')\right)} \\
&= \arg\max_{\underline{y}\in\mathcal{Y}} \exp\left(\underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y})\right) \\
&= \arg\max_{\underline{y}\in\mathcal{Y}_{np}} \underline{w}\cdot\underline{\Phi}(\underline{x},\underline{y}) \\
&= \arg\max_{\underline{y}\in\mathcal{Y}_{np}} \underline{w}\cdot\sum_{(j,k)\in\mathcal{I}} y(j,k)\underline{\phi}(\underline{x},j,k) \\
&= \arg\max_{\underline{s}\in\mathcal{Y}_{np}} \sum_{(j,k)\in\mathcal{I}} y(j,k)\left(\underline{w}\cdot\underline{\phi}(\underline{x},j,k)\right)
\end{aligned}$$

Only change from projective parsing: we've replaced the set of projective parses $\mathcal{Y}$, with the set of non-projective parses, $\mathcal{Y}_{np}$

Decoding in Non-Projective Parsing Models: the Chu-Liu-Edmonds Algorithm

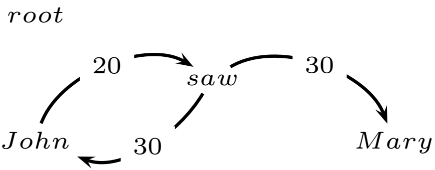


(figure and example from McDonald et al, 2005)

- Goal is to find the highest scoring directed spanning tree

Step 1

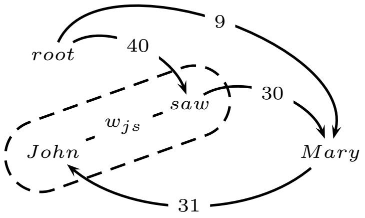
- For each word, find the highest scoring incoming edge:



(figure from McDonald et al 2005)

- If the result of this step is a tree, we have the highest scoring spanning tree
- If not, we have at least one cycle. Next step is to pick a cycle, and *contract* the cycle

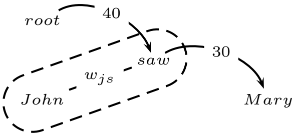
The Result of Contracting the Cycle



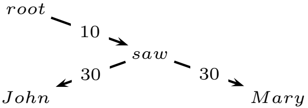
- We merge *John* and *saw* (the words in the cycle) into a single node c
- The weight of the edge from c to *Mary* is 30 (because the weight from *John* to *Mary* is 3, and from *saw* to *Mary* is 30: we take the highest score)
- See McDonald et al 2005 (posted on the class website, under *lectures*) for how the weights from *root* to c and *Mary* to c are calculated
- Having created the new graph, we then recurse (return to step 1)

Step 1 (again)

- For each word, find the highest scoring incoming edge:



- If the result of this step is a tree, we have the highest scoring spanning tree
- This time we have a tree, and we're done (if not, we would repeat step 2 again)
- Retracing the steps taken in contracting the cycle allows us to recover the highest scoring tree:



Efficiency

- A naive implementation takes $O(n^3)$ time (n is the number of nodes in the graph, i.e., the number of words in the input sentence)
- An improved implementation takes $O(n^2)$ time

Estimating the Parameters

- Again, we can choose the parameters that maximize

$$L(\underline{w}) = \sum_{i=1}^{n} \log p(\underline{y}^i | \underline{x}^i; \underline{w}) - \frac{\lambda}{2} ||\underline{w}||^2$$

where $\{(\underline{x}^i, \underline{y}^i)\}_{i=1}^n$ is the training set

- The gradients can again be calculated efficiently (for example, see Koo, Globerson, Carreras, and Collins, EMNLP 2007)