Dependency grammars

Assignment Project Exam Help

- Based on syntactic relations between a **head** word and a **dependent** word
- "Shallower" in hierarchical structure

https://powcoder.com

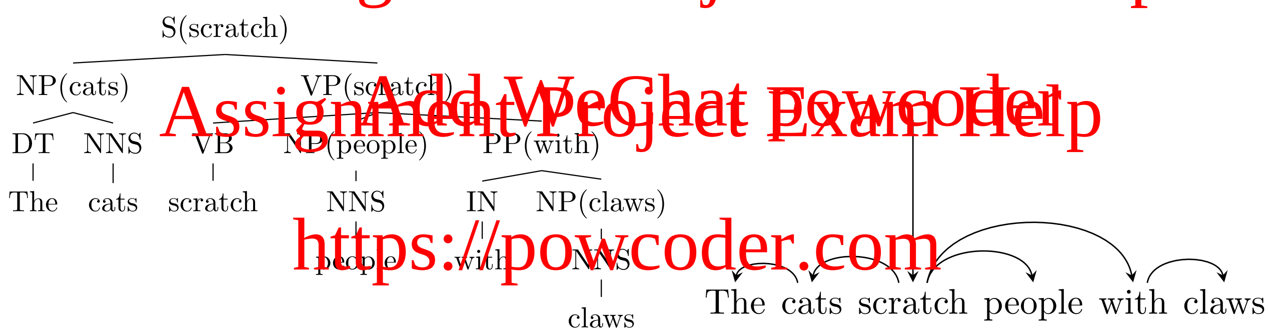
Add WeChat powcoder

Head and dependents

- The head sets the syntactic category of the construction: for example, nouns are the heads of noun phrases, and verbs are the heads of verb phrases.
- The modifier may be optional while the head is mandatory: for example, in the sentence *Cats scratch people with claws*, the subtrees *cats scratch* and *cats scratch people* are grammatical sentences, but *with claws* is not.
- The head determines the morphological form of the modifier: for example, in languages that require gender agreement, the gender of the noun determines the gender of the adjectives and determiners.
- Edges should first connect content words, and then connect function words.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Relationship between phrase structures and dependency structures



(a) lexicalized constituency parse

(b) unlabeled dependency tree

Figure 11.1: Dependency grammar is closely linked to lexicalized context free grammars: each lexical head has a dependency path to every other word in the constituent. (This example is based on the lexicalization rules from § 10.5.2, which make the preposition the head of a prepositional phrase. In the more contemporary Universal Dependencies annotations, the head of *with claws* would be *claws*, so there would be an edge *scratch* → *claws*.)

Labeled dependencies

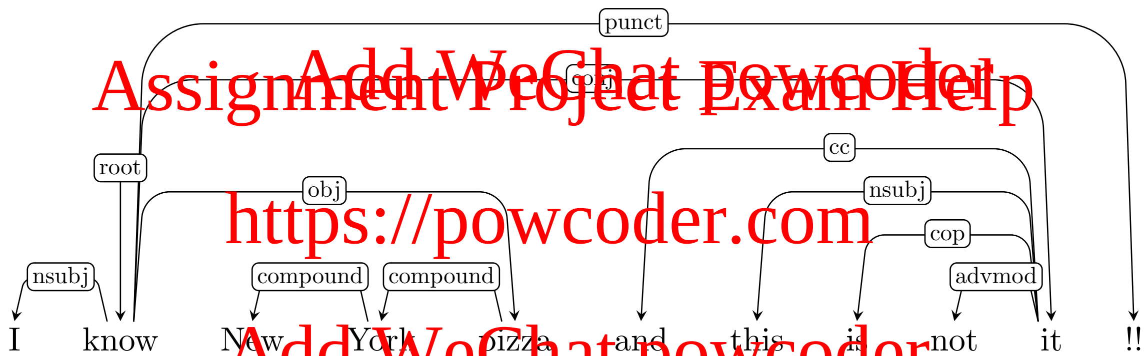


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

Projectivity

- **Projectivity:** An edge from i to j is projective iff all k between i and j are descendants of i. A dependency parse is projective iff all its edges are projective.
- Informally, a dependency parse is projective if there are no crossing edges if all dependencies are drawn on one side of the sentence.

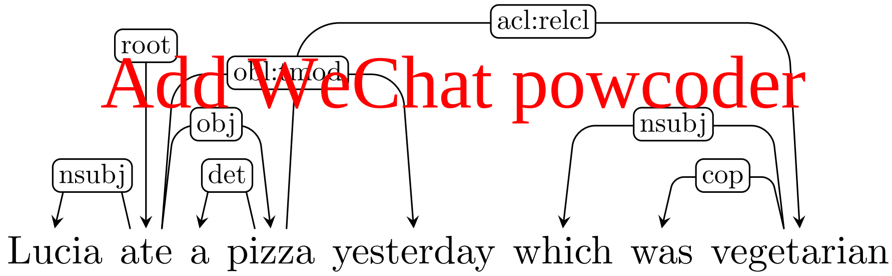


Figure 11.5: An example of a non-projective dependency parse. The "crossing edge" arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

Main dependency parsing approaches

Assignment Project Exam Help

- Graph-based approaches
- Transition-based approaches

https://powcoder.com

Add WeChat powcoder

Graph-based approach

- Let $\boldsymbol{y} = \{(i \xrightarrow{r} j\}$ represent a dependency graph in which r is a relation from headword $i \in \{1, 2, \cdots, M, ROOT\}$ to modifier $j \in \{1, 2, \cdots, M\}$. The special node $ROOT$ indicates the root of the graph, and M is the length of the input $|\boldsymbol{w}|$.
- Given a scoring function $\Psi(\boldsymbol{y}, \boldsymbol{w}; \boldsymbol{\theta})$,

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{w})}{\operatorname{argmax}} \Psi(\boldsymbol{y}, \boldsymbol{w}; \boldsymbol{\theta})$$

  where $\mathcal{Y}(\boldsymbol{w})$ is the set of valid dependency parses on the input $\boldsymbol{w}$.
- The set of possible labels $|\mathcal{Y}(\boldsymbol{w})|$ is exponential in the length of the input.
- Algorithm that search over this space of possible graphs are known as **graph-based dependency parsers**.

Factorization

Dependency parsers that operate under this assumption are called arc-factored.

- First-order factorization:

$$\Psi(\boldsymbol{y}, \boldsymbol{w}; \boldsymbol{\theta}) = \sum_{i \xrightarrow{r} j \in \boldsymbol{y}} \psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta})$$

- Second-order factoriztion:

$$\begin{aligned}\Psi(\boldsymbol{y}, \boldsymbol{w}; \boldsymbol{\theta}) = & \sum_{i \xrightarrow{r} j \in \boldsymbol{y}} \psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta}) \\ & + \sum_{k \xrightarrow{r'} i \in \boldsymbol{y}} \psi_{grandparent}(i \xrightarrow{r} j, k, r', \boldsymbol{w}; \boldsymbol{\theta}) \\ & + \sum_{i \xrightarrow{r'} s \in \boldsymbol{y}, s \neq j} \psi_{sibling}(i \xrightarrow{r} j, s, r', \boldsymbol{w}; \boldsymbol{\theta})\end{aligned}$$

Computing scores for dependency arcs

Assignment Project Exam Help

- Linear $\psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \cdot \boldsymbol{f}(i \xrightarrow{r} j, \boldsymbol{w})$
- Neural $\psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta}) = \text{Feedforward}([\boldsymbol{u}_{w_i}; \boldsymbol{u}_{w_j}]; \boldsymbol{\theta})$
- Generative: $\psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta}) = \log P(w_j, r | w_i)$

https://powcoder.com

Add WeChat powcoder

Linear feature-based arc scores

- The length and direction of the arc;
- The words w_i and w_j linked by the dependency relation;
- The prefixes, suffixes, and parts-of-speech of these words;
- The neighbors of the dependency arc, w_{i-1} , w_{i+1} , w_{j-1} , w_{j+1};
- The prefixes, suffixes, and part-of-speech of these neighbor words.

Learning a linear model with Perceptron

- For a model with feature-based arc scores and perceptron loss, we obtain the usual structured perceptron update
  - Finding the tree with the highest score:

$$\hat{\boldsymbol{y}} = \underset{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{w})}{\operatorname{argmax}} \boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}')$$

  - Update the weights

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \boldsymbol{f}(\boldsymbol{w}, \boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{w}, \hat{\boldsymbol{y}})$$

Learning a linear model with CRF

- A CRF for arc-factored dependency parsing is built on the probability model:

$$P(\boldsymbol{y}|\boldsymbol{w}) = \frac{\exp \sum_{i \xrightarrow{r} j \in \boldsymbol{y}} \psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta})}{\sum_{\boldsymbol{y'} \in \mathcal{Y}(\boldsymbol{w})} \exp \sum_{i \xrightarrow{r} j \in \boldsymbol{y'}} \psi(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta})}$$

  Where the posterior represents the fraction of the exponent of the score of one possible dependency graph out of the sum of the exponent of the scores of all possible graphs

- Questions: How do we compute the score of one dependency graph? How do we compute the sum of scores for all possible graphs?
- Such a model is trained to minimize the negative log conditional-likelihood.

Neural arc scores

- Given vector representations of $\boldsymbol{x}_i$ for each word w_i in the input, a set of arc scores can be computed from a feedforward neural network:

$$\psi(i \xrightarrow{r} j; \boldsymbol{w}, \boldsymbol{\theta}) = \text{FeedForward}([\boldsymbol{x}_i; \boldsymbol{x}_j]; \boldsymbol{\theta}_r)$$

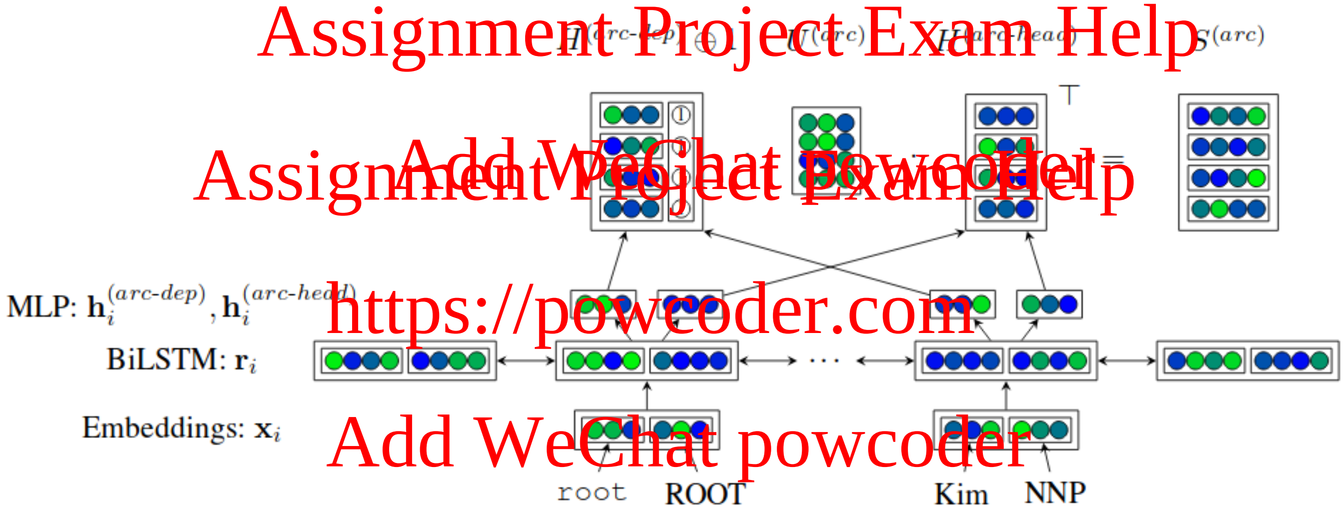
where unique weights $\boldsymbol{\theta}_r$ are available for each arc type.

- Specifically, the score of each arc for each relation can be computed as:

$$\boldsymbol{z} = g(\boldsymbol{\Theta}_r[\boldsymbol{x}_i; \boldsymbol{x}_j] + b_r^{(z)})$$
$$\psi(i \xrightarrow{r} j) = \boldsymbol{\beta}_r \boldsymbol{z} + b_r^{(y)}$$

where $\boldsymbol{\Theta}_r$ is a matrix, $\boldsymbol{\beta}_r$ is a vector, each b_r is a scalar, the function g is an element-wise *tanh* activation function.

SOA in neural dependency parsing: Biaffine models



Tutorial: http://www.cse.chalmers.se/~richajo/nlp2019/l7/Biaffine%20dependency%20parsing.html

Bi-affine models for dependency parsing

- The input is a sequence of word embeddings concatenated with their POS embeddings

$$\boldsymbol{x}_i = \boldsymbol{v}_i^{(word)} \oplus \boldsymbol{v}_i^{(pos)}$$

- The input is fed into a BiLSTM to get a sequence of hidden states

$$\boldsymbol{h}_i, \boldsymbol{c}_i = \mathrm{BiLSTM}((\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n))_i$$

- Each hidden state $\boldsymbol{h}_i$ is projected into four distinct vectors

$$\boldsymbol{h}_i^{(arc-dep)} = \mathrm{MLP}^{(arc-dep}(\boldsymbol{h}_i)$$
$$\boldsymbol{h}_i^{(arc-head)} = \mathrm{MLP}^{(arc-head}(\boldsymbol{h}_i)$$
$$\boldsymbol{h}_i^{(rel-dep)} = \mathrm{MLP}^{(rel-dep}(\boldsymbol{h}_i)$$
$$\boldsymbol{h}_i^{(rel-head)} = \mathrm{MLP}^{(rel-head}(\boldsymbol{h}_i)$$

Predicting the arcs with hidden vectors

- Given a dependent, pair it up with each potential head (all other tokens) in the sentence, and compute a score:

$$\boldsymbol{s}_i^{(arc)} = H^{(arc-head)} W^{(arc)} \boldsymbol{h}_i^{(arc-dep)} + H^{(arc-head)} \boldsymbol{b}^{\top(arc)}$$

- The head is the pair with the highest score

$$y_i'^{(arc)} = \operatorname*{argmax}_j s_{ij}^{(arc)}$$

Predicting relations with hidden states

- Given a head y_i' for word i, perform another biaffine transformation to predict the relation labels
- First use the relation vectors to compute a score for each possible label:

$$
\begin{aligned}
\boldsymbol{s}_i^{(rel)} &= \boldsymbol{h}_{y_i'^{(arc)}}^{\top(rel-head)} \boldsymbol{U}^{(rel)} \boldsymbol{h}_i^{(rel-dep)} \\
&+ W^{(rel)} \left(\boldsymbol{h}_i^{(rel-dep)} \oplus \boldsymbol{h}_{y_i'^{(arc)}}^{(rel-head)} \right) \\
&+ \boldsymbol{b}^{(rel)}
\end{aligned}
$$

  What's the shape of $\boldsymbol{U}$?

- Find the relation label with the highest score:

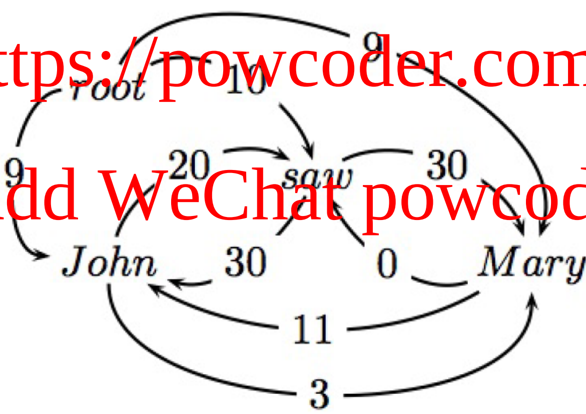
$$
y_i'^{(rel)} = \operatorname*{argmax}_j s_{ij}^{(rel)}
$$

Breaking potential cycles in factored graph-based dependency parsing

- Since in factored graph-based dependency parsing the parent for each dependent is predicted independently by finding the edge with the highest score, it is possible that the resulting dependency structure may have cycles. For instance, the model might be predict i is the head of j, and j is the head of i.
- When this happens, we need to break the cycle to ensure that the resulting dependency tree is well-formed.
- One algorithm that can do this is the Chu-Liu/Edmonds algorithm.

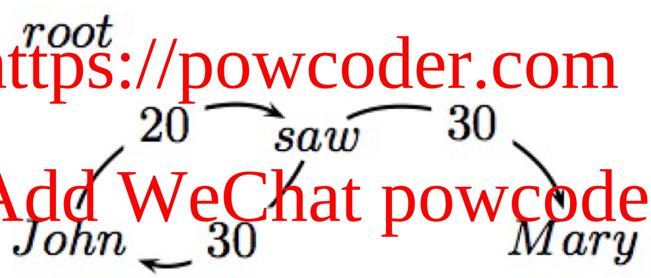
The Chu-Liu-Edmonds algorithm

- Assuming a model that assigns a score to each possible edge in a dependency tree
- x = root John saw Mary



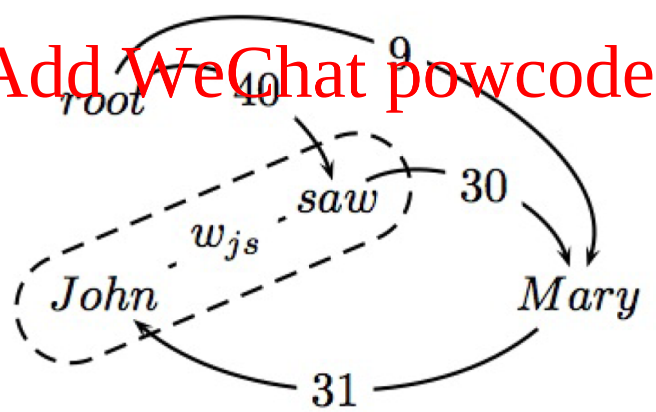
The Chu-Liu-Edmonds algorithm

- start by removing all incoming arcs to *root*, and finding the highest scoring incoming arc for each node



The Chu-Liu-Edmonds algorithm

- If not a tree, identify cycles and contract
- Recalculate arc weights into and out-of cycle
- New incoming arc weights equal to the weight of the best spanning tree that includes head of the incoming arc, and all nodes in cycle
  - root → saw → John: 40
  - root → John → saw: 40



The Chu-Liu-Edmonds algorithm

- This is the final dependency tree: