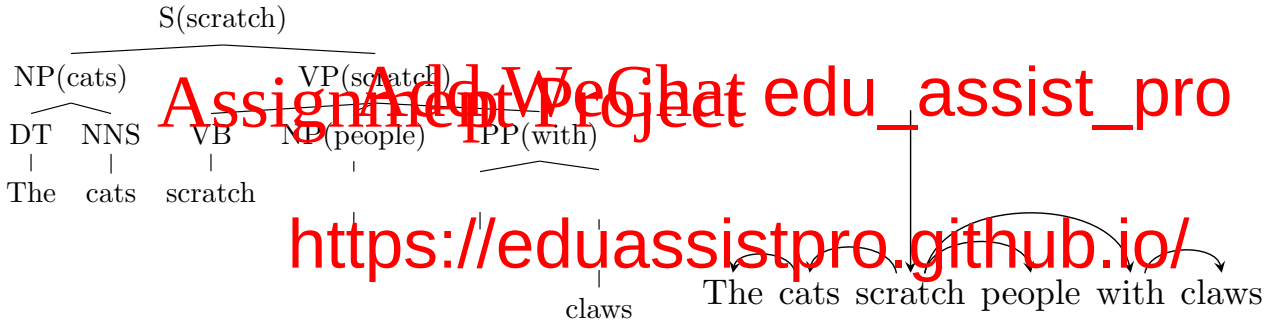
Dependency grammars

- Based on synt word and a **dependent**
- "Shallower

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
  for example, i *e with claws*,
  the subtrees le are
  grammatic
- The head determines the morphologic r:
  for example, in languages that require ge e
  gender of the noun determines the gender of the adjectives and determiners.
- Edges should first connect content words, and then connect function words.

Relationship between phrase structures and dependency structures



(a) lexicalized constituency parse (b) dependency tree

Figure 11.1: Dependency grammar is closely linked to lexi mars: 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

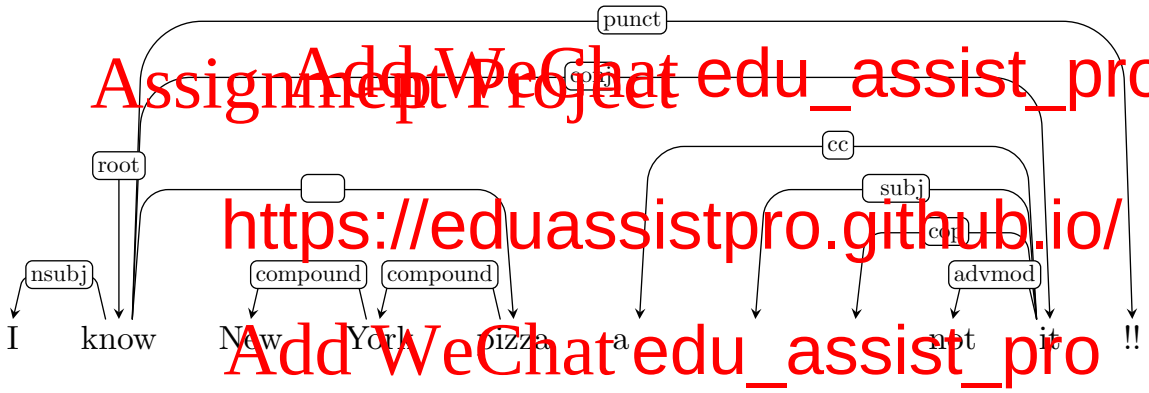


Figure 11.3: A labeled dependency parse from the English UD Treebank (reviews-361348-0006)

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 projectiv crossing edges if all dependencies are drawn o sentence.

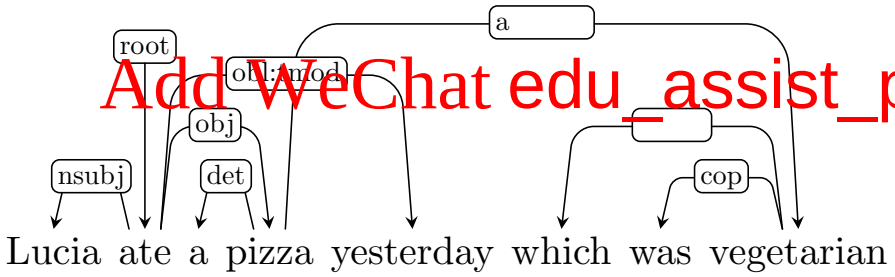


Figure 11.5: An example of a non-projective dependency parse. The "crossing edge" arises from the relative clause *which was vegetarian* and the oblique temporal modifier *yesterday*.

Main dependency parsing approaches

- Graph-base
- Transition-

Graph-based approach

- Let $\boldsymbol{y} = \{(i \xrightarrow{r} j$ h 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 le $|\boldsymbol{w}|$.

- Given a scoring function $Q(\boldsymbol{y}, \boldsymbol{w}; \theta$

$$\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{w})$$

where $\mathcal{Y}(\boldsymbol{w})$ is the set of valid dependenc $\boldsymbol{w}$.

- 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 th
arc-factored.

- First-order factorization:

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

- Second-ord

$$\Psi(\boldsymbol{y}, \sum_{i \xrightarrow{r} j \in \boldsymbol{y}} \psi_{\text{arc}}(i \xrightarrow{r} j, \boldsymbol{w}; \boldsymbol{\theta})$$

$$+ \sum_{k \xrightarrow{r'} i \in \boldsymbol{y}} \psi_{\text{grandparent}}(i \to j, k, r', \boldsymbol{w}; \boldsymbol{\theta})$$

$$+ \sum_{i \xrightarrow{r'} s \in \boldsymbol{y}, s \neq j} \psi_{\text{sibling}}(i \xrightarrow{r} j, s, r', \boldsymbol{w}; \boldsymbol{\theta})$$

Computing scores for dependency arcs

- Linear $\psi(i \quad j, \boldsymbol{w}; \boldsymbol{\theta}) = \boldsymbol{\theta} \quad \boldsymbol{f}(i \quad j$
- Neural $\psi(i$ ard([ ]; $\boldsymbol{\theta}$)

- Generative: $\psi(i \rightarrow j, \boldsymbol{w}; \boldsymbol{\theta}) = \log$

Linear feature-based arc scores

- The length and direction of the arc;
- The words w_i and w_j linked by the dep
- The prefixes, s ds;
- The neighbo , w_{i+1} , w_{j-1} , w_{j+1};
- The prefixes, suffixes, and part-of-spe r words.

Learning a linear model with Perceptron

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

$$\mathbf{y}' \in \mathcal{Y}($$

  - 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:

$$P(\mathbf{y}|\mathbf{w};\boldsymbol{\theta}) = \frac{\exp \sum_r \boldsymbol{\theta}_r f_r(\mathbf{y},\mathbf{w})}{\sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{w})} \exp \sum_r \boldsymbol{\theta}_r f_r(\mathbf{y}',\mathbf{w})}$$

Where the pos nt of the score of one possible dependency gra the exponent of the scores of all possible gra

- 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 repr i w_i in the input, a set of arc scores can be computed from a feedforward neural network:

$$\psi(i \xrightarrow{r} j) = \text{FFNN}([\quad \text{rd}([\quad _r)$$

where uniqu i

- Specifically 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 sequ
with their POS em

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

- The input is fed into a Bi-LSTM to get a sequence o
states

$$\boldsymbol{r}(\boldsymbol{h}_i)$$

- Each hidden state $\boldsymbol{h}_i$ is projected i
ors

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

Predicting the arcs with hidden vectors

- Given a dependent, pair it up with each potenti other tokens) in the sentence, and compute a s

$$s_i^{(arc)} = \qquad\qquad {}^{arc-head)}\boldsymbol{b}^{\top(arc)}$$

- The head is the pair with the highest score

$$y_i^{\prime(arc)} = \arg_j \qquad_{ij}$$

Predicting relations with hidden states

- Given a head https://eduassistpro.github.io/ transformation to predict the relation labels
- First use the relation vectors to compute a score for each Assignment Project Exam Help possible label:

Assignment Project Add WeChat edu_assist_pro

$$s^{(rel)} = h^{T(rel-head)} U^{(re}$$

https://eduassistpro.github.io/

$$i \qquad (arc)^{head)} \Big)$$

Add WeChat edu_assist_pro

$$+ b^{(rel)}$$

What's the shape of U?

- Find the relation label with the highest score:

$$y_i'^{(rel)} = \underset{j}{\text{argmax}}\, s_{ij}^{(rel)}$$

Breaking potential cycles in factored graph-based dependency parsing

- Since in factored graph-based dependenc
  for each dependent is predicted independe
  edge with the highest score, it is possible that th
  dependency
  model might b e
  i. and j is the head of
- When this happens, we need to break the cy
  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 d, 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 weigh spanning tree that includes head of the incomi nodes in cycle
  - root $\rightarrow$
  - root $\rightarrow$

The Chu-Liu-Edmonds algorithm

- This is the final dependency tree.