CSC485/2501 Homework Assignment 1: Write-up

Yinuo Zhao

Student Number: `1007672494` *UTORid:* `zhaoy237`
`inor.zhao@mail.utoronto.ca`

September 30, 2024

1 Transition-based dependency parsing

(a) Complete the sequence of transitions. Please see Table 1 below.

Step	Stack	Buffer	New dep	Transition
⋮	⋮	⋮	⋮	⋮
5	[ROOT, raise, those, doubts]	[is, to, resolve, them]		SHIFT
6	[ROOT, raise, doubts]	[is, to, resolve, them]	those $\xleftarrow{\text{det}}$ doubts	LEFT-ARC
7	[ROOT, raise]	[is, to, resolve, them]	raise $\xrightarrow{\text{dobj}}$ doubts	RIGHT-ARC
8	[ROOT, raise, is]	[to, resolve, them]		SHIFT
9	[ROOT, is]	[to, resolve, them]	raise $\xleftarrow{\text{csubj}}$ is	LEFT-ARC
10	[ROOT, is, to]	[resolve, them]		SHIFT
11	[ROOT, is, to, resolve]	[them]		SHIFT
12	[ROOT, is, resolve]	[them]	to $\xleftarrow{\text{mark}}$ resolve	LEFT-ARC
13	[ROOT, is, resolve, them]	[]		SHIFT
14	[ROOT, is, resolve]	[]	resolve $\xrightarrow{\text{dobj}}$ them	RIGHT-ARC
15	[ROOT, is]	[]	is $\xrightarrow{\text{xcomp}}$ resolve	RIGHT-ARC
16	[ROOT]	[]	ROOT $\xrightarrow{\text{root}}$ is	RIGHT-ARC

Table 1: Sequence of transitions, the first 4 given steps omitted.

(b) Number of Steps. A sentence containing n words will be parsed in **2n** steps. Because:

- SHIFT: every word needs to be moved from the buffer to the stack using the SHIFT operation, this would require **n operations**.
- ARC(LEFT and RIGHT): every word needs to be part of a dependency and needs to be removed from the stack by an ARC operation, this would also require **n operations**.

Therefore, for n words: $n + n = 2n$ operations are needed to parse it.

(g) Comparison with ChatGPT. The prompt used is: `Using transition based parsing, parse the sentence "To ask those questions is to answer them."`
The parsing generated by ChatGPT is:

$$\begin{aligned}
\text{ROOT} &\rightarrow \text{.} \\
\text{is} &\rightarrow \text{ask} \\
\text{ask} &\rightarrow \text{To} \\
\text{ask} &\rightarrow \text{those} \\
\text{ask} &\rightarrow \text{questions} \\
\text{is} &\rightarrow \text{answer} \\
\text{answer} &\rightarrow \text{to} \\
\text{answer} &\rightarrow \text{them}
\end{aligned}$$

The parsing generated from the model is:

$$\begin{aligned}
\texttt{ask} &\rightarrow \texttt{To} \\
\texttt{questions} &\rightarrow \texttt{those} \\
\texttt{ask} &\rightarrow \texttt{questions} \\
\texttt{is} &\rightarrow \texttt{ask} \\
\texttt{answer} &\rightarrow \texttt{to} \\
\texttt{answer} &\rightarrow \texttt{them} \\
\texttt{is} &\rightarrow \texttt{answer}
\end{aligned}$$

The model-generated parsing is more accurate and simpler than the ChatGPT-generated parsing. It correctly attaches the determiner “those” to “questions” and handles the main verb “is” governing both the subject (“ask those questions”) and the predicate (“to answer them”). In contrast, the ChatGPT parsing incorrectly connects “those” to “ask” and complicates the structure by attaching both “ask” and “answer” directly to “is.” ChatGPT’s parsing also treats the period as a separate word which introduces unnecessary complexity, making the model-generated parsing a better representation of the sentence’s syntactic structure.

2 Graph-based dependency parsing

(a) Insufficiency for non-projective dependency trees. Transition-based parsing algorithms and the operations such as shift, left-arc, and right-arc operations, are designed with the assumption that dependencies between words in a sentence can be constructed without crossing other arcs. Dependencies are formed between the **top two items** on the stack. However, this method breaks down in the case of non-projective dependency trees. When dependencies cross, the parser would need to access words that are **deeper in the stack**. As a result, they cannot directly handle non-projective structures without modification or additional mechanisms that allow for more complex, non-local dependencies.

(b) Projective counting results.

```
$ SOLN=1 python3 run_test.py q2
train: 12081/12543 (96.3%)
dev: 1957/2001 (97.8%)
test: 2036/2077 (98.0%)
```

(c) Gap degree. The gap degree for the first tree is 0, because this tree is projective. But we can also compute it as the maximum of the gap degree of each word, which also gives zero, see details as below:

- gap(To) = 0, it has no dependency.
- gap(raise) = 0, dependencies: {To, those, doubts}, "To raise those doubts" has no gap
- gap(those) = 0, it has no dependency.
- gap(doubts) = 0, its dependency is: {those}, the sequence "those doubts" has no gap
- gap(is) = 0, its dependencies are all words in the sentence, thus there are no gaps.
- gap(to) = 0, it has no dependency.
- gap(resolve) = 0, its dependencies are: {to, them}, "to resolve them" has no gap
- gap(them) = 0, it has no dependency.

The gap degree for the second tree is 1, because the maximum gap degree of each words in the sentence is 1, see details below:

- gap(Sam) = 0, it has no dependency.
- gap(met) = 1, dependencies: {Sam, a student, who, has}, there is one gap
- gap(a student) = 1, dependencies: {who, has}, there is one gap

- gap(today) = 0, it has no dependency.
- gap(who) = 0, it has no dependency.
- gap(has) = 0, dependencies: {who, a linguistics degree}, there is no gap
- gap(a linguistics degree) = 0, it has no dependency.

(e) Weight matrices initialization. A uniform distribution between a and b has Mean: $\frac{a+b}{2}$ and Variance: $\frac{(b-a)^2}{12}$.

We want the uniform distribution's mean to be 0:

$$\frac{a+b}{2} = 0 \quad \Rightarrow \quad a = -b$$

We want the variance of the uniform distribution to be $\sigma^2 = \frac{2}{m}$:

$$\frac{(b-a)^2}{12} = \frac{2}{m}$$

Substituting a = -b into this equation:

$$\frac{(b-(-b))^2}{12} = \frac{2}{m} \quad \Rightarrow \quad \frac{(2b)^2}{12} = \frac{2}{m} \quad \Rightarrow \quad \frac{4b^2}{12} = \frac{2}{m} \quad \Rightarrow \quad b^2 = \frac{6}{m} \quad \Rightarrow \quad b = \sqrt{\frac{6}{m}}$$

Since a = -b, we have:

$$a = -\sqrt{\frac{6}{m}} \quad \text{and} \quad b = \sqrt{\frac{6}{m}}$$

(g) Term selection. In arc scoring, the focus is on determining whether a word is a head, which makes the task asymmetric. We include the head-specific term H_A because the structure of the sentence depends primarily on identifying heads, while a dependent's role only makes sense in relation to its head. Adding a dependent-only term D_A wouldn't add meaningful information. In contrast, label scoring is a symmetric task where both the head and dependent contribute equally to classifying the type of dependency relation between them. That's why both H_L and D_L are included in the label scorer.

(h) Double multiplication. Because we are scoring pairs of elements (head and dependent) rather than a single element, as in standard classification tasks. The input is transformed twice because we need to represent both the head and the dependent separately and model the interaction between them. Multiplying the input by weight matrices W_A and W_L captures the contributions of both the head and the dependent to the overall score, allowing us to compute the arc or label scores based on their combined features, which wouldn't be possible with just a single transformation.

(i) Constraints

1. **Constraint 1: No Self-Loops** A word cannot have a dependency arc pointing to itself. This means for any vertex `i` , an arc from vertex `i` to itself is not a valid dependency. In the `mask_possible` function, it is enforced by setting the mask values for all self-referential arcs (positions `(b, i, i, :)` for all batch elements and all dependency relations) to False.
2. **Constraint 2: Root Dependency Relation** Dependency can only exist from the root to other vertices, meaning an arc from the root is allowed, but an arc to the root is not valid. In the `mask_possible` function, it is enforced by setting the mask values for arcs to the root vertex with the root relation (positions `(b, 0, :, 0)`) to False.

(j) MST *vs.* argmax. Because dependency parsing needs to produce a **valid tree structure**, which is a connected acyclic graph. In the context of parsing, this means each word must have exactly one head (root has not head). Maximum spanning tree (MST) algorithm ensures the tree is valid and the scores are maximized. With argmax, these is a possibility that it results in **cycles**(keep cycling to increase the score) and the result is no longer a tree.

(l) Parser mismatch.

- "What if Google expanded on its search-engine (and now e-mail) wares into a full-fledged operating system?"
  - In transition based parsing, the mistakes include labeling “expanded” as an acl instead of advcl, and incorrectly marking “wares” as appos and “system” as nmod instead of their correct relations to “expanded.”
  - In graph based parsing, the key mistake is incorrectly marking “What” as punct instead of the sentence root (root), along with the incorrect obl label for “engine” and mislabeling “e-mail” as nmod instead of part of the conjunction with “search-engine.”
  - I think the graph parsing is slightly better because it correctly marks “wares” and “system” as obl related to “expanded,” but the critical error of mislabeling “What” as punct instead of root still prevents it from being fully accurate.
- "The clerics demanded talks with local US commanders."
  - In transition based parsing, the mistakes include labeling “clerics” as the root and “demanded” as an adjectival modifier (amod), which significantly misrepresents the sentence structure.
  - In graph based parsing, the only mistake is the lack of the more specific nmod:with relation for “commanders,” though the overall structure is correct.
  - I think the graph based parsing is better because it accurately captures the core structure of the sentence, identifying the correct root, subject, and object, whereas the first parsing misidentifies the main verb and root.

- "(And, by the way, is anybody else just a little nostalgic for the days when that was a good thing?)"
  - In transition-based parsing, the mistakes include labeling “And” as the root and incorrectly marking “else” as a parataxis rather than an adjectival modifier of “anybody.”
  - In graph-based parsing, the main mistakes are linking “And” to “days” instead of “nostalgic” and marking “way” as an nmod related to “days” instead of an oblique related to “nostalgic.”
  - Graph based is better because it correctly identifies “nostalgic” as the root and captures key relations like “else” and the relative clause, whereas the first parsing misassigns the root and key dependencies.