Premise:

$$(, i, i), (, i, i) \quad \forall i \in \{0 \dots n\}$$

Rules:

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \ (i \rightarrow j) \quad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \ (j \rightarrow i) \quad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)} \quad \forall i < k \leq j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)} \quad \forall i \leq k < j$$

Goal:

$$(, 0, n)$$

This note describes a new algorithm for sentence compression using dependency parsing. We begin by giving notation for standard dependency parsing and then introduce our extensions.

1 First-Order Dependency Parsing

Each item for standard dependency parsing consists of a tuple (t, i, j) where t is a symbol in $\{, , , \}$ and $0 \leq i \leq j \leq n$.

The first group of rules create attachments and the second complete the items

2 Skip Bigram Parsing

We extend the first order model to skip parsing to additionally score the bigrams chosen in the final parse. To do this we extend the item definition to include the anticipated next word (t, i, j).

The only new addition is that we use the "hook trick" to select the best next word for each premise item before using it. The other rules are all identical.

Note that for this to work, it is crucial that we only allow skipping words on the right and that the left side index i is always the left-most word used in the item.

This parsing algorithm has runtime $O(n^3)$.

Premise:

$$(, i, i), (, i, i) \quad \forall i \in \{0 \dots n\}$$

Rules:

$$\frac{(, i, i)}{(, i, j)} \text{ Bigram}(i, j+1) \quad \text{(hook trick)} \qquad \forall 0 \leq i \leq j \leq n$$

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \qquad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \qquad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)} \qquad \forall i < k \leq j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)} \qquad \forall i \leq k < j$$

Goal:

$$(, 0, n, n+1)$$

3 Second-Order Dependency Parsing

In second-order dependency parsing we want to score not only single arcs like $i \rightarrow j$ but to also take into account the previous (sibling) modifier. Second-order arcs take the form $i \rightarrow k\ j$.

The full dynamic program for second-order parsing requires adding a new type informally called "box" - . A box type maintains two indices k and j that will eventually be modifiers to the same head i; however we don't yet know what that head will be.

The completion rules are identical to first-order parsing.

Premise:

$$(, i, i), (, i, i) \quad \forall i \in \{0 \dots n\}$$

Rules:

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \quad \forall i \leq k < j$$

$$\frac{(, i, i) \quad (, i+1, j)}{(, i, j)}\ (i \rightarrow j) \quad \forall i < j$$

$$\frac{(, i, j-1) \quad (, j, j)}{(, i, j)}\ (j \rightarrow i) \quad \forall i < j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)}\ (i \rightarrow k\ j) \quad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)}\ (j \rightarrow k\ i) \quad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)} \quad \forall i < k \leq j$$

$$\frac{(, i, k) \quad (, k, j)}{(, i, j)} \quad \forall i \leq k < j$$

Goal:

$$(, 0, n)$$

4 Second-Order Skip Bigram

For second-order skip bigram parsing we introduce another one more type . The semantics of this type "has not yet taken modifiers". It's only use is for creating arcs without siblings ij. In standard second-order parsing we did not need to remember this piece of information because it was implicit in the fact that the arc had index i, i. However with skip words this needs to be marked on right items.

Premise:

$$(, i, i, i), (, i, i, i) \quad \forall i \in \{0 \dots n\}$$

Rules:

$$\frac{(, i, i)}{(, i, j)} \text{ Bigram}(i, j+1) \text{ (hook trick)} \qquad \forall 0 \leq i \leq j \leq n$$

$$\frac{(, i, j)}{(, i, j)} \quad \forall 0 \leq i \leq j \leq n+1$$

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \qquad \forall i \leq k < j$$

$$\frac{(, i, k) \quad (, k+1, j)}{(, i, j)} \ (i \rightarrow j) \qquad \forall i < j$$

The rest of the rules are identical to second-order parsing.

5 Computing with Fixed m

Finally we discuss how to find a compression with a fixed value of m. In each of the systems presented we do not keep track of the total number of words used.

We propose doing this using the following Lagrangian relaxation. Say the constrained version of the optimization is to find

$$\max_y \quad f(y) \quad s.t.$$
$$\sum_{i=1}^{n} y_i = m$$

The Lagrangian dual of this problem is

$$\begin{aligned} L(\lambda) &= \max_y f(y) - \lambda(\sum_{i=1}^{n} y_i - m) \\ &\max_y (f(y) - \lambda \sum_y y_i) + \lambda m \end{aligned}$$

In order to compute the score we need to incorporate these multipliers in the problem. However this can easily be added to the rules that incorporate the bigrams.

$$\frac{(, i, i)}{(, i, j)} \text{ Bigram}(i, p) \quad \text{(hook trick)} \qquad \forall i, k, 0 \leq i < j \leq n+1$$

Each time we apply this rule we add in a λ term to the score of the bigram.

The dual problem is to find

$$\min_\lambda L(\lambda)$$

However since we only have one variable, we can minimize this by bisection (binary search).