Sentence Compression as Dependency Parsing

First Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Second Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Abstract

1 Introduction

2 Related Work

(?)

(?)

3 Background

We begin by presenting formal background for the structures underlying our approach for sentence compression. Our system models sentence compression as a joint parsing and language modeling problem. We first describe the specifics of these techniques and then their combined use.

Given an input sentence $x_1 \dots x_n$, define the set of feasible compressions as all ordered subsets, i.e. a compression is $x_{p_1} \dots x_{p_m}$ for a sequence p with $1 \leq p_1 < \dots < p_m \leq n$ and $m < n$. The length of the compression is m.

Our aim is the find the highest-scoring compression under a given objective.

Define the set of possible bigrams $\mathcal{Y}$ as

$$\mathcal{I} = \big\{(i,j) : 0 \leq i < j \leq n+1\big\}$$

$$\mathcal{Y} = \begin{cases} \sum_{i=1}^{n+1} y(0,i) = 1, \sum_{i=0}^{n} y(i,n+1) = 1 \\ \sum_{i=0}^{j-1} y(i,j) = \sum_{k=j+1}^{n+1} y(j,k) \end{cases}$$

The score of a bigram sequence is $\theta \in R^{\mathcal{I}}$

Define the set of dependency arcs as

$$\mathcal{J} = \big\{(i,j) : \forall\, i \in \{0 \dots n\}, j \in \{1 \dots n\}\big\}$$

A projective dependency parse is a vector in y in $\mathcal{Y}$ where $\mathcal{Y} \subset \{0,1\}^{\mathcal{I}}$ where

$$\mathcal{Z} = \{z \in \{0,1\}^{\mathcal{J}} : z \text{ forms a directed tree } \}$$

Note that in standard dependency parsing, we require this to be a full directed spanning tree. For this work we only require that it be a tree. For a parse in $y \in Y$ define the span of y as $||y||_1 = \sum_{h,m} y(h,m)$. Standard dependency parsing requires the span of a parse to be n.

The score of a dependency sequence is $\omega \in R^{\mathcal{J}}$

Our aim to find the highest-scoring combination of dependency parse score and language model score.

$$\begin{aligned} \max_{y \in \mathcal{Y}, z \in \mathcal{Z}} \quad & \theta^{\top} y + \omega^{\top} z \\ \text{s.t.} \quad & \sum_{i=0}^{j-1} y(i,j) = \sum_{i=0}^{n} z(i,j) \;\; \forall 1 \leq j \leq n \end{aligned}$$

The find the highest-scoring structure in this objective. However, past work has shown that this problem is NP-hard (). Some success has been shown for using ILP solvers () and dual decomposition ().

In this paper we focus on a variant of this problem where we further assume that $\mathcal{Z}$ is *projective*. Projective dependency parsing is widely used in the parsing literature. Informally, this means that for any $z(i,j) = 1$ and $z(i',j') = 1$ the arcs do not cross each other.

With this assumption we can show that finding the highest-scoring compression can be found in much more efficiently.

4 Decoding

We now present the decoding algorithm for this problem. We begin by reviewing the standard decoding algorithm for projective dependency parsing, and then give the extension for this problem.

4.1 Eisner's Algorithm

4.2 Skip-Word Dependency 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 $O(n^3)$.

5 Extensions

5.1 Span Requirements

For compression problems we often have a target size of the compressed sentence. That is we are given a ratio r such that our target sentence should have $n/r = M$ words. We can incorporate this constraint directly into our combinatorial optimization problem

$$q(M) = \max_{z \in \mathcal{Z}} \theta^\top y + \omega^\top z \text{s.t}$$
$$\sum_{h=0:h\neq m}^{n} y(h,m) = \sum_{j=m+1}^{n+1} y(m,j)$$
$$\sum_{(h,m)\in\mathcal{I}} y(h,m) = M$$

This extra requirement can be represented by a single constraint. However in practice it is difficult to incorporate this into the dynamic programming algorithm. One idea is to intersect with a counting FSA. In general each item may span d vertices with $0 \leq d \leq M$, and each sub-item may span d' and $d - d'$ vertices respectively, i.e.

$$\frac{(\square, i, k, d') \quad (\triangle, k, j, d)}{(\triangle, i, j, d)} \quad \forall i < k \leq j, 0 \leq d' < d \leq< M$$

This intersection adds a factor of M^2 to the running time, which makes the full algorithm $O(n^5)$ which is intractable in practice.

An alternative approach is to relax the single span constraint.

```
procedure BISECTION($M$)
    $\lambda^{(\uparrow)} \leftarrow \max_{(i,j)\in\mathcal{J}} \omega(i,j)$
    $\lambda^{(\downarrow)} \leftarrow -\max_{(i,j)\in\mathcal{J}} \omega(i,j)$
    $\lambda^{(0)} \leftarrow \frac{\lambda^{(\uparrow)}+\lambda^{(\downarrow)}}{2}$
    for $k = 1$ to $K$ do
        $y^{(k)}, z^{(k)} =_{y,z} L(\lambda^{(k-1)}, y, z)$
        if $||y^{(k)}||_1 = M$ then
            return $y^{(k)}, z^{(k)}$ $||y^{(k)}||_1 > M$
            $\lambda^{(\uparrow)} \leftarrow \lambda^{(k)}$
            $\lambda^{(k+1)} \leftarrow \frac{\lambda^{(k)}+\lambda^{(\downarrow)}}{2}$ $||y^{(k)}||_1 < M$
            $\lambda^{(\downarrow)} \leftarrow \lambda^{(k)}$
            $\lambda^{(k+1)} \leftarrow \frac{\lambda^{(\uparrow)}+\lambda^{(k)}}{2}$
        end if
    end for
end procedure
```

$$L(\lambda, y, z) = \theta^\top y + \omega^\top z + \lambda(||y||_1) - M\lambda \text{s.t}$$
$$\sum_{h=0:h\neq m}^{n} y(h,m) = \sum_{j=m+1}^{n+1} y(m,j)$$

For any fixed value of λ we can calculate $y_\lambda, z_{\lambda y,z} L(\lambda, y, z)$ using the dynamic programming algorithm shown. We simply replace θ with θ' where $\theta'(h,m) = \theta(h,m) + \lambda$ for all $(h,m) \in \mathcal{I}$.

By weak duality $\theta^\top y_\lambda + \omega^\top z_\lambda$ will always give an upper bound on the optimal constrained solution y^*, z^*. And if $||y_\lambda||_1 = M$ then this upper bound is provably tight. Past work in NLP has looked at minimizing this dual upper bound with subgradient descent; however we can exploit the fact that there is only a single dual variable λ and use a more efficient method.

We will minimize $L(\lambda)$ using the bisection method. First note that with $M = n$ we can solve $q(m)$ by setting λ to the max bigram score , similarly with $M = 0$ we can solve $q(0)$ by setting λ to the inverse of the max bigram score.

The bisection method is guaranteed to find an optimal solution if one exists in

$$iterations$$

[redacted]

5.2 Second-Order Parsing

The same method extends to second-order dependency parsing. For a second-order parser we replace the index set $\mathcal{I}$ with a new index set that also includes a *sibling* index.

$$\mathcal{I} = \begin{cases} (h, s, m) : & h \in \{0 \dots n\}, m \in \{1 \dots n\}, h \neq m, \\ & h < s < m \text{ or } m < s < h \text{ or } s = \epsilon \end{cases}$$

Second order parsing also require $O(n^3)$ time.

The main complication is that standard second-order parsing includes a rule of the form

$$\frac{(\text{◺}, i, i) \quad (\text{◿}, i+1, j)}{(\text{⏢}, i, j)} \quad \forall i < j$$

for constructing an arc $y(i, \epsilon, j)$. This rule relies on an implicit property that $(\text{◺}, i, i)$ is the only right-facing item that has not yet taken a modifier.

We replace these rules with

$$\frac{(\text{◺⋱}, i, k) \quad (\text{◿}, k+1, j)}{(\text{⏢}, i, j)} \quad \forall i \leq k < j$$

where the symbol ◺⋱ implies that the word s_i has not yet taken a modifier and that the words s_{i+1} through s_k have been "skipped".

Besides this change the rules are identical to standard second-order parsing.

6 Training

7 Features

8 Experiments

Premises:

$$(\text{◿}, i, i), (\text{◺}, i, i) \;\; \forall\, 0 \le i \le j \le n$$

Goal:

$$(\text{◺}, 0, n)$$

Completion Rules:

$$\frac{(\text{⏢}, i, k) \quad (\text{◺}, k, j)}{(\text{◺}, i, j)} \qquad \forall\, 0 \le i < k \le j \le n$$

$$\frac{(\text{◿}, i, k) \quad (\text{⏥}, k, j)}{(\text{◿}, i, j)} \qquad \forall\, 0 < i \le k < j \le n$$

First-Order Rules:

$$\frac{(\text{◺}, i, k) \quad (\text{◿}, k+1, j)}{(\text{⏢}, i, j)} \; z(i, j) = 1 \qquad \forall\, 0 \le i \le k < j \le n$$

$$\frac{(\text{◺}, i, k) \quad (\text{◿}, k+1, j)}{(\text{⏥}, i, j)} \; z(j, i) = 1 \qquad \forall\, 0 < i \le k < j \le n$$

Second-Order Rules:

$$\frac{(\text{◺}, i, k) \quad (\text{◿}, k+1, j)}{(\square, i, j)} \qquad \forall\, 0 < i \le k < j \le n$$

$$\frac{(\text{◺}, i, i) \quad (\text{◿}, i+1, j)}{(\text{⏢}, i, j)} \; z(i, \epsilon, j) \qquad \forall\, 0 \le i < j \le n$$

$$\frac{(\text{◺}, i, j-1) \quad (\text{◿}, j, j)}{(\text{⏥}, i, j)} \; z(j, \epsilon, i) \qquad \forall\, 0 < i < j \le n$$

$$\frac{(\text{⏢}, i, k) \quad (\square, k, j)}{(\text{⏢}, i, j)} \; z(i, k, j) \qquad \forall\, i \le k < j$$

$$\frac{(\square, i, k) \quad (\text{⏥}, k, j)}{(\text{⏥}, i, j)} \; z(j, k, i) \qquad \forall\, 0 \le i \le k < j \le n$$

Revised Premises:

$$(\text{◿}, i, i), (\text{◣}, i, i) \;\; \forall\, 0 \le i \le j \le n$$

Additional Rules:

$$\frac{(\text{◣}, i, j)}{(\text{◺}, i, j)} \; y(i, j+1) = 1 \qquad \forall\, 0 \le i \le j \le n$$

$$\frac{(\text{◣}, i, i)}{(\text{◣}, i, j)} \qquad \forall\, 0 \le i < j \le n$$

$$\frac{(\text{◣}, i, k) \quad (\text{◿}, k+1, j)}{(\text{⏢}, i, j)} \; y(i, k+1), z(i, \epsilon, j) \qquad \forall\, 0 \le i < k < j \le n \;\; \text{[2nd-Order only]}$$

Figure 1: Deductive rules for skip-bigram parsing.