SkipDep Compression

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 Background

Our aim is to simultaneously maximize

Given a sentence $s_1 \dots s_n$ we would like to find the maximum ski-dependency parse for a sentence.

Define the set of dependency arcs as

$$\mathcal{I} = \{(h, m) : \forall\, h \in \{0 \dots n\}, m \in \{1 \dots n\}, h \neq m\}$$

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

$$\mathcal{Y} = \text{formsadirectedtree}$$

Note that in standard dependency parsing, we require this to a 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.

Define the set of possible bigrams as

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

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

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

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

Define the score of a dependency arc as

$$\theta(, h, m) = w \cdot \phi(s, h, m)$$

maybe the bigram scores come from a language model.

$$\omega(i, j) = \log p(s_i, s_j)$$

3 Decoding Algorithm

We now present the decoding algorithm for this problem. The main idea is quite simple. We extend the standard Eisner parsing algorithm to allow each premises to "skip" words.

3.1 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 $O(n^3)$.

4 Extensions

4.1 Span Requirements

For compression problems we often have a target size of the compressed sentence. That is we

Premise:

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

Rules:

$$\frac{(\triangle, i, i)}{(\triangle, i, j)}\ z(i, j+1) = 1 \qquad \forall 0 \le i \le j \le n$$

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

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

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

$$\frac{(\square, i, k) \quad (\triangle, k, j)}{(\triangle, i, j)} \qquad \forall i < k \le j$$

$$\frac{(\triangle, i, k) \quad (\square, k, j)}{(\triangle, i, j)} \qquad \forall i \le k < j$$

Goal:

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

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 $0 \le d \le M$, and each sub-item may span d' and $d - d'$ vertices respectively, i.e.

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

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.

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

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.

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

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

$$iterations$$

[redacted]

4.2 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.

$$\mathcal{I} = \{\,(h, s, m) : h \in \{0 \dots n\}, m \in \{1 \dots n\}, h \neq m, h < s < m \text{ or } m < s < h \text{ or } s = \text{NULL}\}$$

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

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

$$\frac{(\triangleright, i, i) \quad (\triangleleft, i+1, j)}{(\square, i, j)} \quad \forall i < j$$

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

We replace these rules with

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

where the symbol $\triangleright$ 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.

Premise:

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

Rules:

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

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

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

$$\frac{(\triangleright, i, k) \quad (\triangleleft, k+1, j)}{(\square, i, j)}\ y(i, \text{NULL}, j) \qquad \forall i < j$$

$$\frac{(\triangleright, i, j-1) \quad (\triangleleft, j, j)}{(\square, i, j)}\ y(j, \text{NULL}i) \qquad \forall i < j$$

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

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

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

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