Personal Project: Shift-Reduce Dependency Parsing

1 Problem Statement

The goal of this project is to implement a shift-reduce dependency parser. This entails two subgoals:

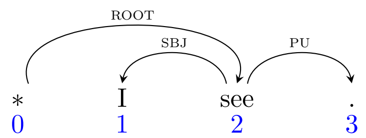
- *Inference*: We must have a shift-reduce parser that finds the correct parse given an oracle.
- *Learning*: We must choose a model that approximates the oracle and train it with labeled data.

2 Framework

2.1 Projective Dependency Trees

Given a sentence $\underline{x} = x_1 \ldots x_n$, we want to find its dependency tree structure. The n words $x_1 \ldots x_n$ correspond to n nodes in the tree. A valid dependency tree $\underline{y}$ for $\underline{x}$ is a directed tree over the n nodes rooted at a special node $*$. We will characterize it as a set of arcs (i, j, l) where the pair (i, j) forms a directed edge and $l \in \mathcal{L}$ is the label of the edge. An example is worth a thousand words:

$$\underline{x} = \text{I} \quad \text{see} \quad .$$
$$\underline{y} = \{(0, 2, \text{ROOT}), (2, 1, \text{SBJ}), (2, 3, \text{PU})\}$$



We will only consider *projective* dependency trees, in which for every edge (i, j) there is a directed path from i to all nodes between i and j. As an illustration, tree (a) below is not projective since there is no path from 3 to 4 even though there is an edge $(3, 5)$. In contrast, tree (b) is projective.

(a) (b)

A projective dependency tree has the "nested property" that for every word x, all words that are reachable from x form a contiguous subsequence of the sentence. Note that tree (a) violates this property.

2.2 Shift-Reduce Parsing

At any point in parsing sentence $\underline{x}$, a shift-reduce parser maintains a **parser configuration** $c = (S, Q, A)$ with respect to $\underline{x}$ where

- S is a stack $[\ldots i]_S$ of nodes that are processed.
- Q is a queue $[j \ldots]_Q$ of nodes that are yet to be processed.
- A is a set of arcs at this point.

The parser moves from one configuration to the next by performing one of the following transitions:

- `left-arc`(l): $([\ldots i, j]_S, Q, A) \Rightarrow ([\ldots j]_S, Q, A \cup \{(j, i, l)\})$
- `right-arc`(l): $([\ldots i, j]_S, Q, A) \Rightarrow ([\ldots i]_S, Q, A \cup \{(i, j, l)\})$
- `shift`: $([\ldots]_S, [i \ldots]_Q, A) \Rightarrow ([\ldots i]_S, [\ldots]_Q, A)$

Let $\mathcal{T}$ be the set of all transitions. Given $\underline{x} = x_1 \ldots x_n$, the parser initializes the configuration as $c = ([0]_S, [1 \ldots n]_Q, \{\})$ and applies a sequence of transitions $t \in \mathcal{T}$ to reach the goal configuration $c = ([0]_S, []_Q, A)$ for some final set of arcs A. Then it returns $\underline{y} = A$ as the predicted dependency tree.

3 Inference

Let o be an oracle that predicts the correct transition $o(\underline{x}, c) = t \in \mathcal{T}$ for any configuration c with respect to sentence $\underline{x}$. Using this oracle, we can find the true dependency tree for any sentence with the algorithm **Shift-ReduceParse**. Note that the running time is linear in the length of the sentence n, since there can be at most $2n$ transitions before reaching the goal configuration.

Shift-ReduceParse
Input: a sentence $\underline{x} = x_1 \ldots x_n$, an oracle o
Output: a dependency tree $\underline{y}$, transition history H

- Initialize $c \leftarrow ([0]_S, [1 \ldots n]_Q, \{\})$ and $H \leftarrow \{\}$.
- While $|S| > 1$ or $Q \neq []$,
  - ◇ $t \leftarrow o(\underline{x}, c)$
  - ◇ $H \leftarrow H \cup \{(c, t)\}$
  - ◇ $c = (S, Q, A) \leftarrow t(c)$
- Return $\underline{y} = A$ and H.

4 Learning

Given Q training examples $(\underline{x}^{(1)}, \underline{y}^{(1)}) \ldots (\underline{x}^{(Q)}, \underline{y}^{(Q)})$ where $\underline{x}^{(q)}$ is a sentence and $\underline{y}^{(q)}$ is the dependency tree associated with it, we want to train a predictor $\hat{o}$ that approximates the oracle o.

4.1 Sample Extraction

Since the oracle receives a sentence $\underline{x}$ and a parser configuration c with respect to $\underline{x}$ as the input and returns a transition $t \in \mathcal{T}$ as the output, we need to prepare samples of form $((q, c), t)$ where $q \in [Q]$ points to the relevant sentence $\underline{x}^{(q)}$. For this purpose, we make use of an auxiliary function **NextTransition**.

NextTransition
Input: a dependency tree $\underline{y}$, a configuration $c = (S, Q, A)$
Output: the next transition to be applied to c based on $\underline{y}$

1. Return `shift` if $|S| < 2$.
2. Otherwise, $S = [\ldots i, j]_S$ for some $i < j$.
   (a) Return `left-arc`(l) if $(j, i, l) \in \underline{y}$.
   (b) Return `right-arc`(l) if $(i, j, l) \in \underline{y}$ and every $(j, j', l') \in \underline{y}$ is also in A.
   (c) Return `shift` otherwise.

The extra condition in 2(b) makes sure that node j parents all its children before it is removed from the stack. This is not necessary in 2(a) since if the tree $\underline{y}$ is projective, node i must parent all its children before reaching j in order to satisfy the nested property.

Now we can extract a set of samples $E = \{((q^{(z)}, c^{(z)}), t^{(z)})\}_{z=1}^{Z}$ of some size Z using the algorithm **ExtractSamples**.

ExtractSamples
Input: training examples $(\underline{x}^{(1)}, \underline{y}^{(1)}) \ldots (\underline{x}^{(Q)}, \underline{y}^{(Q)})$
Output: a set of samples $E = \{((q^{(z)}, c^{(z)}), t^{(z)})\}_{z=1}^{Z}$

- $E \leftarrow \{\}$
- For $q = 1 \ldots Q$,
  - ◇ Define oracle o_q for $\underline{x}^{(q)}$ as follows. Given configuration c, the oracle will predict

    $$o_q(\underline{x}^{(q)}, c) = \textbf{NextTransition}(\underline{y}^{(q)}, c)$$

  - ◇ $\underline{y}_q, H_q \leftarrow$ **Shift-ReduceParse**$(\underline{x}^{(q)}, o_q)$ // $\underline{y}_q = \underline{y}^{(q)}$ must hold
  - ◇ $E \leftarrow E \cup \{((q, c), t) : (c, t) \in H_q\}$
- Return E.

4.2 Feature Representation

Now that we have labeld samples $((q, c), t)$, it is straightforward to train a multiclass classifier that mimics the oracle. But first, we must decide on how to represent the input (q, c). Let ϕ be a feature function that maps a sentence-configuration pair $(\underline{x}, c)$ to a d-dimensional vector $\phi(\underline{x}, c) \in \mathbb{R}^d$. We can use any features in $\underline{x}$ and $c = (S, Q, A)$ useful for making prediction, such as

- Part-of-speech tags of the nodes on the stack
- Word identities of the nodes on the stack
- Labels of the arcs originating from the nodes on the stack

For example, suppose we extract a sample $((q, c), t)$ where

$$\begin{aligned} \underline{x}^{(q)} &= \text{I} \quad \text{see} \quad . \\ c &= ([0, 2, 3]_S, []_Q, \{(2, 1, \text{SBJ})\}) \\ t &= \texttt{right-arc}(\text{PU}) \end{aligned}$$

We can use a binary vector $v = \phi(\underline{x}^{(q)}, c) \in \mathbb{R}^d$ to encode the following information:

$$\begin{aligned} \text{ROOT} &= \text{True} \\ \text{POS}(3) &= \text{SYM} \\ \text{POS}(2) &= \text{VB} \\ \text{WORD}(3) &= . \\ \text{WORD}(2) &= \text{see} \\ \text{ARC-L}(3) &= \varnothing \\ \text{ARC-R}(3) &= \varnothing \\ \text{ARC-L}(2) &= \text{SBJ} \\ \text{ARC-R}(2) &= \varnothing \end{aligned}$$

For notational cleanness, we will use $E' = \{(v^{(z)}, t^{(z)})\}_{z=1}^{Z} = \{(\phi(\underline{x}^{(q^{(z)})}, c^{(z)}), t^{(z)})\}_{z=1}^{Z}$ to denote the set of feature-transformed samples.

4.3 Averaged Perceptron

A linear classifier keeps a weight vector $w_t \in \mathbb{R}^d$ for each $t \in \mathcal{T}$ and defines a score function $f(w_t, \phi(c)) \in \mathbb{R}$. Given a sentence $\underline{x}$ and a parser configuration c with respect to $\underline{x}$, an oracle approximator $\hat{o}$ using this classifier will predict

$$\hat{o}(\underline{x}, c) = \arg\max_{t \in \mathcal{T}} f(w_t, \phi(\underline{x}, c))$$

We will choose the averaged perceptron as our classifier, which defines $f(w_t, \phi(c)) = w_t \cdot \phi(c)$. The weight vector $w_t \in \mathbb{R}^d$ is learned from feature-transformed samples $E' = \{(v^{(z)}, t^{(z)})\}_{z=1}^{Z}$ using the algorithm **TrainAveragedPerceptron**. Two remarks on this specific installment of the algorithm:

- *Averaging*: Instead of storing a vector $w_t^{r,z} \in \mathbb{R}^d$ for all $t \in \mathcal{T}, r \in [R], z \in [Z]$ and then averaging

$$w_t = \frac{\sum_{r=1}^{R} \sum_{z=1}^{Z} w_t^{r,z}}{RZ}$$

  we keep distinct weights w_t' only once and record how many examples it endures without making a mistake by a dictionary s_t. Then the final weights are given by

$$w_t \leftarrow \frac{\sum_{w_t' \in s_t} w_t' \times s_t(w_t')}{\sum_{w_t' \in s_t} s_t(w_t')}$$

- *Update*: The update scheme here is called "ultraconservative": $w_t \leftarrow w_t + \gamma_t \phi(c^{(z)})$ where $\gamma_{t^{(z)}} = 1$, $\sum_{t \neq t^{(z)}} \gamma_t = -1$, and $\gamma_t = 0$ for $t \in \mathcal{T}$ on which no mistake is made. The normalization contraint is necessary for the percetron convergence guarantee.

TrainAveragedPerceptron
Input: $E' = \{(v^{(z)}, t^{(z)})\}_{z=1}^{Z}$, number of rounds $R \in \mathbb{N}$
Data Structure: a dictionary s_t for each $t \in \mathcal{T}$
Output: $w_t \in \mathbb{R}^d$ for each $t \in \mathcal{T}$

- $w_t \leftarrow (0, \ldots, 0) \in \mathbb{R}^d$ for all $t \in \mathcal{T}$
- For $r = 1 \ldots R$, for $z = 1 \ldots Z$,
  - ◇ For $t \in \mathcal{T}$, $s_t(w_t) \leftarrow s_t(w_t) + 1$ if $w_t \in s_t$, $s_t(w_t) \leftarrow 1$ otherwise
  - ◇ Find a set of transitions that incorrectly scored higher than the true transition:

$$\Psi = \{t \in \mathcal{T} - \{t^{(z)}\} : w_t \cdot v^{(z)} > w_{t^{(z)}} \cdot v^{(z)}\}$$

  - ◇ If $|\Psi| > 0$,
    - ∗ Update $w_{t^{(z)}} \leftarrow w_{t^{(z)}} + v^{(z)}$
    - ∗ For $t \in \Psi$, update $w_t \leftarrow w_t - \frac{1}{|\Psi|} v^{(z)}$
- Return

$$w_t \leftarrow \frac{\sum_{w_t' \in s_t} w_t' \times s_t(w_t')}{\sum_{w_t' \in s_t} s_t(w_t')} \text{ for all } t \in \mathcal{T}$$

4.4 Summary

Here we summarize the procedure of estimating the oracle developed in this section. The inputs are training data of dependency trees, a feature function ϕ to represent any sentence-configuration pair $(\underline{x}, c)$, and the number of training rounds R.

EstimateOracle
Input: $(\underline{x}^{(1)}, \underline{y}^{(1)}) \ldots (\underline{x}^{(Q)}, \underline{y}^{(Q)})$, feature function ϕ, number of rounds R
Output: an oracle appoximator $\hat{o}$

- $E \leftarrow$ **ExtractSamples**$((\underline{x}^{(1)}, \underline{y}^{(1)}) \ldots (\underline{x}^{(Q)}, \underline{y}^{(Q)}))$
- $E' \leftarrow \{(\phi(\underline{x}^{(q)}, c), t) : ((q, c), t) \in E\}$
- $\{w_t\}_{t \in \mathcal{T}} \leftarrow$ **TrainAveragedPerceptron**(E', R)
- Return an oracle approximator $\hat{o}$ that predicts for any sentence $\underline{x}$ and a configuration c with respect to $\underline{x}$

$$\hat{o}(c) = \arg\max_{t \in \mathcal{T}} w_t \cdot \phi(\underline{x}, c)$$

References

Sandra Kübler, Ryan McDonald, and Joakim Nivre (2009). Dependency Parsing.

Joakim Nivre (2005). Inductive Dependency Parsing of Natural Language Text.

Massimiliano Ciaramita and Giuseppe Attardi (2007). Dependency Parsing with Second-Order Feature Maps and Annotated Semantic Information.

Michael Collins (2002). Discriminative Training Methods for HMMs: Theory and Experiments with Perceptron Algorithms.