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 $x = x_1 \ldots x_n$, 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 $y$ for $x$ 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:

$$\begin{align*}
x &= I \text{ see } . \\
y &= \{(0, 2, \text{ROOT}), (2, 1, \text{SBJ}), (2, 3, \text{PU})\}
\end{align*}$$

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 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 $x$, a shift-reduce parser maintains a **parser configuration** $c = (S, Q, A)$ with respect to $x$, 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**: \([\ldots i]_S, Q, A \Rightarrow ([\ldots i]_S, Q, A)\)

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

### 3 Inference

Let \(o\) be an oracle that predicts the correct transition \(o(x, c) = t \in T\) for any configuration \(c\) with respect to sentence \(x\). 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 \(2^n\) transitions before reaching the goal configuration.

**Shift-ReduceParse**

**Input**: a sentence \(x = x_1 \ldots x_n\), an oracle \(o\)

**Output**: a dependency tree \(y\), transition history \(H\)

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

### 4 Learning

Given \(Q\) training examples \((x^{(1)}, y^{(1)}), \ldots, (x^{(Q)}, y^{(Q)})\) where \(x^{(q)}\) is a sentence and \(y^{(q)}\) 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 \(x\) and a parser configuration \(c\) with respect to \(x\) as the input and returns a transition \(t \in T\) as the output, we need to prepare samples of form \(((q, c), t)\) where \(q \in [Q]\) points to the relevant sentence \(x^{(q)}\). For this purpose, we make use of an auxiliary function **NextTransition**.

**NextTransition**

**Input**: a dependency tree \(y\), a configuration \(c = (S, Q, A)\)

**Output**: the next transition to be applied to \(c\) based on \(y\)

1. Return **shift** if \(|S| < 2\).
2. Otherwise, \(S = [\ldots i, j]_S\) for some \(i < j\).
   - (a) Return **left-arc** \((l)\) if \((j, i, l) \in y\).
   - (b) Return **right-arc** \((l)\) if \((i, j, l) \in y\) and every \((j, j', l') \in y\) 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 $y$ 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 $(\mathbf{x}^{(1)}, y^{(1)}) \ldots (\mathbf{x}^{(Q)}, y^{(Q)})$ |
| 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 $\mathbf{x}^{(q)}$ as follows. Given configuration $c$, the oracle will predict $o_q(\mathbf{x}^{(q)}, c) = \text{NextTransition}(y^{(q)}, c)$
  - $y_q, H_q \leftarrow \text{Shift-ReduceParse}(\mathbf{x}^{(q)}, o_q)$  // $y_q = y^{(q)}$ must hold
  - $E \leftarrow E \cup \{(q, c, t) : (c, t) \in H_q\}$
- Return $E$.

### 4.2 Feature Representation

Now that we have labeled 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 $\phi$ be a feature function that maps a sentence-configuration pair $(\mathbf{x}, c)$ to a $d$-dimensional vector $\phi(\mathbf{x}, c) \in \mathbb{R}^d$. We can use any features in $\mathbf{x}$ 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

$\mathbf{x}^{(q)} = \text{I see }$.  
$c = ([0, 2, 3], \emptyset, \{(2, 1, \text{SBJ})\})$

$t = \text{right-arc(PU)}$

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

- $\text{ROOT} = \text{True}$
- $\text{POS(3)} = \text{SYM}$
- $\text{POS(2)} = \text{VB}$
- $\text{WORD(3)} = .$
- $\text{WORD(2)} = \text{see}$
- $\text{ARC-L(3)} = \emptyset$
- $\text{ARC-R(3)} = \emptyset$
- $\text{ARC-L(2)} = \text{SBJ}$
- $\text{ARC-R(2)} = \emptyset$

For notational cleanness, we will use $E' = \{v^{(z)}, t^{(z)}\}_{z=1}^{Z} = \{\phi(\mathbf{x}^{(z)}, c^{(z)})\}_{z=1}^{Z}$ 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 $x$ and a parser configuration $c$ with respect to $x$, an oracle approximator $\hat{o}$ using this classifier will predict

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

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 $\gamma_t = 1$, $\sum_{t \not\in \mathcal{T}} \gamma_t = -1$, and $\gamma_t = 0$ for $t \in \mathcal{T}$ on which no mistake is made. The normalization contraint is necessary for the perceptron 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 $t \in \mathcal{T}$, $r \in \{1 \ldots R\}$, $z \in \{1 \ldots Z\}$, |
|   - 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 $w_t \leftarrow \frac{\sum_{w_t' \in s_t} w_t' \times s_t(w_t')}{\sum_{w_t' \in s_t} s_t(w_t')}$ 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 $\phi$ to represent any sentence-configuration pair $(x, c)$, and the number of training rounds $R$. 

4
EstimateOracle

**Input:** \((x^{(1)}, y^{(1)}), \ldots, (x^{(Q)}, y^{(Q)})\), feature function \(\phi\), number of rounds \(R\)

**Output:** an oracle approximator \(\hat{o}\)

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

\[\hat{o}(c) = \arg \max_{t \in T} w_t \cdot \phi(\bar{x}, c)\]

References

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

