L&S II: Assignment 4

Prof. Noah Smith

Due: Thursday, November 16 (hardcopy, in class)

1 Dynamic Programming Algorithm

A head automaton (HA) is like a finite-state automaton that generates a pair of sequences, $\langle \alpha, \beta \rangle \in (\Sigma^*)^2$. Often HAs are used to define dependency grammars. For example, a HA is defined for each word in the vocabulary Σ; whenever a symbol $s \in \Sigma$ is generated, its HA is called to generate $\langle \alpha, \beta \rangle$. Then α becomes the sequence of left children of s and β its sequence of right children. In general, the two sequences α and β may not be independent of each other.

Formally, a weighted head automaton H is a tuple $\langle \mathcal{Q}, \Sigma, \mathcal{I}, \mathcal{F}, \delta \rangle$ where $\mathcal{Q}$ is a finite set of states, $\mathcal{I} \subseteq \mathcal{Q}$ is the set of initial states, $\mathcal{F} \subseteq \mathcal{Q}$ is the set of final states, and δ is a nondeterministic function that assigns weights to the transitions:

$$\delta : (\mathcal{Q} \setminus \mathcal{F}) \times \Sigma \times \{\leftarrow, \rightarrow\} \times \mathcal{Q} \rightarrow \mathbb{R} \tag{1}$$

H generates $\langle \alpha, \beta \rangle$ by following transitions from state to state. Each transition results in the output of a symbol in Σ along with an arrow ($\leftarrow$ or $\rightarrow$). A $\leftarrow$ means that the symbol is appended to the left end of α, and a $\rightarrow$ means that the symbol is appended to the right end of β. Each transition incurs a real-valued weight; the score of a derivation is the sum of weights of the transitions crossed. (Note that $\delta(q, t, q', \leftarrow) = -\infty$ means that there is no transition from q to q' that appends t to α—or equivalently that the transition has infinite cost.) In this way, α is generated from right to left, and β is generated from left to right. The formal derivation relation $\Rightarrow$ for H is such that:

$$\begin{aligned}
\langle q, v, \alpha, \beta \rangle &\Rightarrow \langle q', v + \delta(q, t, \rightarrow, q'), \alpha, \beta t \rangle & (2)\\
\langle q, v, \alpha, \beta \rangle &\Rightarrow \langle q', v + \delta(q, t, \leftarrow, q'), t\alpha, \beta \rangle & (3)\\
\langle \alpha, \beta \rangle \in L(H) \quad &\text{iff} \quad \exists q_0 \in \mathcal{I}, \exists q_f \in \mathcal{F}, \exists v > -\infty : \langle q_0, 0, \epsilon, \epsilon \rangle \Rightarrow^* \langle q_f, v, \alpha, \beta \rangle, & (4)
\end{aligned}$$

Exercise 1 Suppose you are given H and a pair of sequences $\langle \alpha, \beta \rangle$. Let $\alpha = \langle \alpha_m, \alpha_{m-1}, ..., \alpha_1 \rangle$ and let $\beta = \langle \beta_1, \beta_2, ..., \beta_n \rangle$. Your job is to write recursive equations defining a dynamic programming algorithm that finds the **best path** (the one with the highest weight) through H that generates $\langle \alpha, \beta \rangle$, if it exists. What are the asymptotic runtime and space requirements of your algorithm?

Exercise 2 Suppose that H is guaranteed to be "split." This means[1] that there exists a single state $q_{\text{split}} \in \Omega$ such that all paths go through q_{split} exactly once, all transitions *before* q_{split} are of the $\rightarrow$ variety, and all transitions *after* q_{split} are of the $\leftarrow$ variety. Informally, this means β is generated completely before α is generated. Does this improve the runtime and/or space requirements of your algorithm? If not, can you give an improved algorithm for this special case?

Exercise 3 (bonus) How would you modify your algorithms in exercises 1.1 and 1.2 to *count* paths rather than find the best path?

2 Loss Functions

For each of the following measures of accuracy, show how it can be described as a **factored** loss function. That is, give the complete definition of a loss function that, if minimized, would be equivalent to obtaining maximum *accuracy* (as defined in each case). You will need to define some notation to make your answer clear!

- For a POS tagger, the fraction of words correctly tagged (as compared to a gold standard).
- For a dependency parser, the fraction of words whose parent is correctly identified (as compared to a gold standard). Assume the root word is linked to an invisible parentless parent, \$, at the left edge of the sentence.
- For a word aligner that links words in English to their corresponding words in French (given parallel text), the fraction of links that are correct (i.e., in a gold standard).
- For a named entity recognizer, the fraction of named entities (contiguous subsequences of a sentence) that are correctly bracketed (as compared to a gold standard). You may assume that named entities are not recursive in the gold standard or the hypothesis.

3 Designing a Linear Program

A linear program in standard form looks like this:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c} \cdot \mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Let N be the dimensionality of $\mathbf{x}$ and let M be the number of linear constraints (not including the positivity constraints); $\mathbf{A} \in \mathbb{R}^{M \times N}$ and $\mathbf{b} \in \mathbb{R}^{M}$. It is not terribly difficult to show that *any* linear program can be transformed to this form.

[1] In one definition!

You are encouraged at this juncture to take a careful look at the Klein-Taskar tutorial on maximum margin methods at ACL 2005, linked from the course web page.

Exercise 1 Let $\langle w_0 = \$, w_1, w_2, ..., w_n\rangle$ be a sentence. Let $cost(w_i, w_j)$ be the cost associated with making w_i the parent of w_j, and let the cost of a tree be $\sum_j cost(w_{\text{parent}(j)}, w_j)$. Your job is to define an LP such that the minimizing $\mathbf{x}$ can be converted back into a dependency tree such that

- every word in $\langle w_1, ...w_n\rangle$ has exactly one parent in $\{w_0, w_1, ..., w_n\}$;
- w_0 does not have a parent;
- the solution to the LP will correspond to the lowest-cost dependency tree.

To answer the question, you must define $\mathbf{x}$ (in terms of the dependency tree), $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$. Hint: you should be able to define a solution in which $N = O(n^2)$ and $M = O(n)$.

Note: do not worry about cyclicity yet.

Exercise 2 Do exercise 1 again, but this time add an overall **projectivity** constraint on the tree:

- if w_i is the parent of w_j, then $\forall k \in (i, j) \cup (j, i)$, w_k is a descendent of w_i.

Hint: this will require at least another $O(n^2)$ linear constraints.

Note: do not worry about cyclicity yet.

Exercise 3 (bonus) Give the dual of the LP in exercise 3.1.

Exercise 4 (bonus) The dependency structures recovered above have not been constrained to be *acyclic*. In practice we typically want dependency structures to be trees. Can you define constraints that will enforce cyclicity? You may introduce more variables, too, if you need to. You may also use non-linear constraints.