Assignment 4

CSE 447 and 517: Natural Language Processing – University of Washington

Winter 2021 – Due: February 19, 2021, 11:59 pm

Submit: Your submission will be through Gradescope and will have two parts:

- Writeup for all problems *except* problem 1. Your writeup for the non-implementation problems will be turned in separately as a pdf with one-inch margins and 11-point Times font. Part of the training we aim to give you in this class includes practice with technical writing. Organize your report as neatly as possible, and articulate your thoughts as clearly as possible. We prefer quality over quantity.
- Problem 1. In a zip file (this is the only acceptable format!), you will submit your output (formatted just like the input), your code, a neatly written README file to instruct how to run your code with different settings, and a pdf report containing your answers this problem's questions only. We assume that you always follow good practice of coding (commenting, structuring), and these factors are not central to your grade. You may implement the models in the programming language of your choice. However, please provide well commented code if you want partial credit. If you have multiple files, please provide a short description in the preamble of each file. Do not flood the report with tangential information such as low-level documentation of your code that belongs in code comments or the README. Similarly, when discussing the experimental results, do not copy and paste the entire system output directly to the report. Instead, create tables and figures to organize the experimental results.

1 CSE 447 and CSE 517 Students: Implementation Problem

This tarball includes two files:

- `15pctmasked.txt` contains single sentences, one per line (i.e., newline characters delimit sentences), each with some of its characters "masked"; characters are delimited by whitespace, masked characters are replaced with "<mask>", spaces are denoted "<s>", and the end-of-sequence symbol is "<eos>"
- `lm.txt` contains a bigram language model over characters; the format on each line is "a space b tab $p(b \mid a)$" with special space and end-of-sequence characters denoted as above.

Your job is to decode the sentences, replacing each "masked" character with a character from the bigram model's vocabulary. For input $\boldsymbol{x}$, your output $\boldsymbol{y}$ should be the sequence that maximizes $p(\boldsymbol{y})$ under the bigram language model, subject to the constraint that, for every position i where $\boldsymbol{x}$ is *not* masked, $y_i = x_i$. (In plain English, we want you to find the most likely unmasked sequence consistent with the input, under the model we gave you.) Hint: you should adapt the Viterbi algorithm to solve this problem exactly in polynomial time and space.

Your output file should be in the same format as the input, but with every mask token replaced as described above. Your writeup should explain how you generate a score for each possible choice and how you recover the best sequence. You may use existing libraries and tools for this problem (be sure to acknowledge them in your solution), but we do not expect they will be very helpful.

2 CSE 447 and CSE 517 Students: Based on Eisenstein 9.3 (p. 213)

Construct (i.e., draw on the page; a TA recommends this tool) a finite-state automaton in the style of figures 9.2 and 9.3 (pp. 187–8), which handles the following examples:

- *nation*/N, *national*/ADJ, *nationalize*/V, *nationalizer*/N
- *America*/N, *American*/ADJ, *Americanize*/V, *Americanizer*/N

Be sure that your FSA does not accept any further derivations, such as **nationalizeral* and **Americanizern*.

3 CSE 447 and CSE 517 Students: Viterbi for Second-Order Models

At the end of the CRF lecture (slide 106), we proposed a second-order sequence model that defines scores of *triplets* of labels, $s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2})$. The problem to be solved when we decode with this model is:

$$\hat{\boldsymbol{y}} = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{L}^\ell} \sum_{i=0}^{\ell-1} s(\boldsymbol{x}, i, y_i, y_{i+1}, y_{i+2}) \tag{1}$$

Note that there are ℓ terms in the sum. y_0 (which is a special START label) appears only in the first term ($i = 0$) and $y_{\ell+1}$ (which is a special STOP label) appears only in the last term ($i = \ell - 1$). In a conventional trigram-style model, we might also have "$s(\boldsymbol{x}, -1, y_{-1} = \text{START}, y_0 = \text{START}, y_1)$" in the summation, but including such a term doesn't add any expressive power to what is written above.

The Viterbi algorithm can be adapted for this model. The form of the algorithm is nearly identical to what we saw in lecture. First, prefix scores are computed "left to right," each using a formula with a local max, and along with each prefix score we also store a backpointer that stores the "argmax." Then the backpointers are followed "right to left" to select a label for each word in turn. Here is most of the algorithm:

Input: scores $s(\boldsymbol{x}, i, y'', y', y), \forall i \in \langle 0, \ldots, \ell \rangle, \forall y'' \in \mathcal{L}, \forall y' \in \mathcal{L}, \forall y \in \mathcal{L}$

Output: $\hat{\boldsymbol{y}}$

1. Base case: $\heartsuit_2(y', y) = s(\boldsymbol{x}, 0, \text{START}, y', y)$
   (Note that the subscript in "$\heartsuit_2$" refers to the last position in the triple, here, the word labeled "y," while the "0" argument to s refers to the first position in the triple.)
2. For $i \in \langle 3, \ldots, \ell - 1 \rangle$:
   - $\boxed{\text{Solve for } \heartsuit_i(*, *) \text{ and } b_i(*, *).}$
3. Special case for the end (note a small correction in this line relative to an earlier version of the assignment, which included an equality on the left to "$\heartsuit_{\ell-1}(y', y)$" that should not have been there):

$$\heartsuit_\ell(y', y) = s(\boldsymbol{x}, \ell - 1, y', y, \text{STOP}) + \underbrace{\max_{y'' \in \mathcal{L}} s(\boldsymbol{x}, \ell - 2, y'', y', y) + \heartsuit_{\ell-1}(y'', y')}_{b_\ell(y', y) \text{ is the "argmax"}}$$

   The above looks a little different from what you saw in lecture, where we calculated "$\heartsuit_{\ell+1}(\text{STOP})$." Writing it as we've done here is just a slightly different choice that gives you a little more of a hint. To see why, take a close look at slide 70.
4. $(\hat{y}_{\ell-1}, \hat{y}_\ell) \leftarrow \operatorname*{argmax}_{y', y \in \mathcal{L}^2} \heartsuit_\ell(y', y)$
5. For $i \in \langle \ell - 2, \ldots, 1 \rangle$:
   - $\hat{y}_i \leftarrow b_{i+2}(\hat{y}_{i+1}, \hat{y}_{i+2})$

Deliverables:

1. Give the recurrence for $\heartsuit_i(y', y)$ and for $b_i(y', y)$ (the boxed bit of the algorithm above). We've given you the base case and the special case for the end of the sequence.
2. Analyze the runtime and space requirements of this algorithm as functions of $|\mathcal{L}|$ (the number of labels) and ℓ (the length of the input sequence $\boldsymbol{x}$ and the output sequence $\hat{\boldsymbol{y}}$).

4 CSE 517 Students: Introducing PCFGs

In this problem, you'll become acquainted with **probabilistic context-free grammars** (PCFGs), a more powerful family of models that can be used to parse sequences of words into trees. Chapters 9 and 10 in the textbook give extensive motivation for these kinds of models, as well as algorithms for their use. This problem will reveal a connection between PCFGs and HMMs.
A PCFG is defined as:

- A finite set of "nonterminal" symbols $\mathcal{N}$
  - A start symbol $S \in \mathcal{N}$
- A finite alphabet Σ, called "terminal" symbols, distinct from $\mathcal{N}$
- Production rule set $\mathcal{R}$, each of the form "$N \rightarrow \boldsymbol{\alpha}$" where
  - The lefthand side N is a nonterminal from $\mathcal{N}$
  - The righthand side $\boldsymbol{\alpha}$ is a sequence of zero or more terminals and/or nonterminals: $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$
    * Special case: **Chomsky normal form** constrains $\boldsymbol{\alpha}$ to be either a single terminal symbol or two nonterminals
- For each $N \in \mathcal{N}$, a probability distribution over the rules where N is the lefthand side, $p(* \mid N)$. We will use p_r to denote the conditional probability corresponding to rule $r \in \mathcal{R}$.

A PCFG defines the probability of a tree $\boldsymbol{t}$ as:

$$p(\boldsymbol{t}) = \prod_{r \in \mathcal{R}} p_r^{\mathrm{count}_{\boldsymbol{t}}(r)} \tag{2}$$

where $\mathrm{count}_{\boldsymbol{t}}(r)$ is the number of times rule r is used in the tree $\boldsymbol{t}$.
The definition of an HMM was given in class; we spell it out more carefully here.

- A finite set of states $\mathcal{L}$ (often referred to as "labels")
  - A probability distribution over initial states, p_{start}
  - One state denoted ⯃ is the special stopping state
- A finite set of observable symbols $\mathcal{V}$
- For each state $y \in \mathcal{L}$:
  - An "emission" probability distribution over $\mathcal{V}$, $p_{emission}$
  - A "transition" probability distribution over next states, $p_{transition}$

An HMM defines the probability distribution of a state sequence $\boldsymbol{y}$ and an emission sequence $\boldsymbol{x}$ (length n, excluding the stop symbol) as shown on slide 36 in the lecture.

1. Demonstrate how to implement any HMM using a PCFG. Do this by showing formally how to construct each element of the PCFG ($\mathcal{N}$, S, Σ, $\mathcal{R}$, and p) from the HMM. Hint: think of a state sequence as a right-branching tree; the nonterminals will correspond to HMM states/labels, and you will probably want to add a new nonterminal to serve as the start state S.

2. Transform the grammar you defined above into Chomsky normal form. Every tree derived by the original grammar must have a corresponding tree in the new grammar, with the same probability. Hint: his might be easier if you add a special "start" word to the beginning of every sequence recognized by the new grammar, and you will likely need to add some new nonterminals, as well. On page 202 of your textbook, the transformation is sketched out for arbitrary context-free grammars that are not probabilistic; to fully complete this exercise, you need to handle the probabilities as well as the rules, but only need to show how to do it for your HMM-derived PCFG. Try to give a more precise definition of the transformation than is given in the textbook.

3. Explain how to transform your new PCFG's derivations back into derivations under the original PCFG.