CS 753: Automatic Speech Recognition
Assignment #2 (35 points)

Instructor: Preethi Jyothi*

March 24, 2024

Instructions: This assignment is due on or before **11.59 pm on April 14, 2024**. The submission portal on Moodle will be closed after midnight on April 14.

- This is a group assignment. One submission can be made per team.
- For this assignment, you will run all your code via a Colab notebook. **Important:** While submitting, please make sure that your submission includes fully run notebooks; do not clear the outputs. This will make it easier to cross-check what the TAs see at runtime with what you got.
- Submit a tgz file named `assgmt2.tgz` on Moodle. It should contain a `README.txt` that contains the names and roll numbers of all the team members. Other things that need to be added to `README.txt` appear in Part II of the assignment. Any other notes that you would like to convey to your TAs can be added to `README.txt` as well. Maintain the following directory structure within `assgmt2.tgz`:

```
+- README.txt
+- assgmt2-partI.pdf
+- assgmt2-partII.ipynb
+- constrainedbeamsearch.pdf
+- assgmt2-extracredit.ipynb [OPTIONAL]
```

  - For Part I, submit your solutions neatly written (preferably typed) in `assgmt2-partI.pdf`.
  - All the submitted code will be in a single Colab notebook `assgmt2-partII.ipynb`.
  - `constrainedbeamsearch.pdf` is an additional document that you need to submit for Part II(C).

*Special thanks to Darshan Prabhu, and TAs Dhiraj Sah and Vaibhav Raj who helped set up the coding part of this assignment.

Part I: HMMs and WFSTs **[20 points]**

(A) Viterbi Variants **[10 points]**

Consider an HMM $\lambda = (A, B)$ with a sequence of hidden states Q, a sequence of observations O, transition probabilities $a_{ij} = \Pr(q_t = j | q_{t-1} = i)$ and emission probabilities $b_j(o_t) = \Pr(o_t | q_t = j)$.

1. Efficient Viterbi. Given an HMM and a sequence of observations $O = o_1, \ldots, o_T$, the most probable sequence of states $q_1, \ldots, q_T$ can be efficiently computed using the Viterbi algorithm in $O(N^2 T)$ time where N is the size of the state space. Let us consider a specific HMM where $a_{ii} = p$, $a_{ij} = q \; \forall j \neq i$ and $p > q$. How can we modify the Viterbi algorithm such that it runs in $O(NT)$ time? [**3 points**]

2. k-repeat Viterbi. Modify the recurrence for $v_t(j)$ (i.e. the probability of being in state j after seeing the first t observations) in the original Viterbi algorithm for HMMs so that it finds the best sequence among all sequences in which there is at least one run of k consecutive occurences of the same state. [**3 points**]

3. Extended HMMs. In the HMMs we studied in class, a single observation is emitted on reaching a state. For problems like speech recognition, it could be beneficial to consider HMMs where a sequence of ℓ observations can be emitted on transitioning into a particular state. While one could model this in a standard HMM using a sequence of states to emit the observations one-by-one, this results in HMMs that are exponentially large in ℓ. If the probability distribution of the emitted sequences can be compactly represented, a more efficient representation would be to extend the HMM model to allow emitting sequences in a single time step. Specifically, we define an *Extended HMM* as one in which the observation probability distribution is replaced by a pair of functions L, B, where $L(i, \ell) = \Pr(\ell \mid i)$ is the probability of choosing a length $\ell \geq 1$ for the emitted sequence at state i, and $B(i, o_1, \ldots, o_\ell) = \Pr(o_1, \ldots, o_\ell \mid i, \ell)$ is the probability of emitting a specific ℓ-long sequence $o_1, \ldots, o_\ell$ on reaching state i, conditioned on the sequence being of length ℓ. We will assume that for all i, $L(i, \ell) = 0$ for $\ell > \ell_{\text{max}}$.

In this problem, you need to develop a Viterbi-style algorithm for extended HMMs. That is, given an extended HMM and a sequence of observations $o_1, \ldots, o_T$, you should find the most probable sequence of states $q_1, \ldots, q_m$, $m \leq T$. Define a recursive function $v_t(j)$ which computes the probability of the most likely state sequence that ends in state j while emitting the observation sequence $o_1, \ldots, o_t$. [**4 points**]

(B) Syllables in WFST-based ASR **[10 points]**

Syllables are sub-word units consisting of multiple phones. In this problem, a syllable is defined as a sequence of phones that appear in the order VC, CV or CVC (where V stands for a vowel and C stands for a consonant). We shall assume that our lexicon consists only of words that have at least one way of being written as a sequence of such syllables (e.g., the lexicon will have no word with 3 consecutive consonants, or a word that is just a single vowel). E.g., the words "up", "shoe" and "tall" are composed of single syllables of the form VC, CV and CVC, respectively, and the word "codas" could be broken up either as "cod-as" (CVC, VC) or "co-das" (CV, CVC).

Recall that a WFST-based ASR decoder uses the composed machine, $H \circ C \circ L \circ G$. We want to modify the components H, C, L and G in order to work with syllables instead of individual phones. The HMMs that constitute H now correspond to syllables rather than phones (i.e., the output alphabet of H is the set of all syllables). We wish to still use a phone-based pronunciation model. However, we shall require that words respect syllable boundaries. E.g., the phrase "tap in" can be broken up as "tap-in" (CVC, VC) but not as "ta-pin" (CV, CVC), as in the latter a syllable ("pin") straddles two words.

1. Defining C and L. How should C and L be defined? Draw state diagrams of C and L assuming there are only two phones c and v, a consonant and a vowel, three syllables $\alpha = vc$, $\beta = cv$ and $\gamma = cvc$, and three words x, y and z with pronunciations $cvcv$, $vcvc$ and $cvcvc$, respectively. C and L may accept empty strings. Your solution should readily generalize to larger alphabets. [**4 points**]

2. Contextualized syllables. We define a sequence of "contextualized syllables" to be a sequence of pairs of the form (S, P), where S is a syllable and P is a *phone* following it (P is empty if S is the last syllable in an utterance). For instance, the syllable sequence "re-map-it" would correspond to the contextualized syllable sequence "(re, m), (map, i), (it, ϵ)."

Suppose that we are given an H that outputs contextualized syllables, and L, G are as in a phone-based model. For this problem, we allow syllables to straddle word boundaries. Then, how should C be defined? Draw a state diagram of C assuming the same phone and syllable alphabets as in part (a). [**6 points**]

Part II: Hinglish ASR **[15 points + extra credit]**

For this part, you will work with OpenAI's Whisper-small model to recognize code-switched Hindi-English speech (i.e. speech utterances predominantly containing Hindi, with some English words scattered within).

Code template. You will work with the following Colab notebook where we will explore various decoding techniques.

> **Note:** All the decoding routines in this notebook can run on a CPU, if you're unable to access the GPU. Naturally, this will come with an overhead. Greedy decoding takes ≈ 5 minutes on the GPU and ≈ 35 minutes on the CPU, while beam decoding takes ≈ 10 minutes on the GPU and ≈ 2 hours on the CPU.

(A) Zero-shot Greedy Decoding **[1 point]**

In the notebook, run all the steps from the start until (and including) Task 2.1. This loads a pretrained Whisper-small model and runs greedy decoding for 61 utterances in the `test` split of the code-switched Hinglish corpus (available at this link). This zero-shot evaluation will give you a character error rate (CER) of **0.66-0.69** and word error rate (WER) of **0.87-0.89** on the test set.

Next, load a Whisper-small model that has been finetuned on a small amount of labeled Hinglish speech; it is available at: `https://www.cse.iitb.ac.in/~pjyothi/cs753/whisper-small-finetuned.pt`. What is the new test CER and WER you get using the fine-tuned model? Add a new cell to your notebook at the end of Task 1 (and before Task 2 begins) to load the model and compute these values. Also write down the CER/WER values in `README.txt`.

> **Note:** We fine-tuned the Whisper-small model on the `train` subset of the Hinglish corpus (for 30 training epochs) using the fine-tuning scripts in the following Colab notebook. Go through the code in this notebook. This know-how can come in handy for the extra credit part.

(B) Constrained Filtering-based Greedy Decoding **[4 points]**

Vanilla greedy decoding greedily picks the best token at each decoding step. Here is an alternate sampling technique. We first filter the distribution of logits to only retain the top-N tokens. From within these top-N, we pick the most probable tokens until their cumulative sum is less than a predefined threshold p. Sample one of these tokens, after the top-N filtering and p thresholding, based on the underlying distribution as the chosen token at each decoding step. Implement the function `sample_batch` that is defined in Task 2.2. As mentioned in the comments, you can define any new helper functions that you call from within `sample_batch`.

What is the new test CER and WER you get using the fine-tuned model and this constrained greedy decoding with $N = 10$ and $p = 0.9$? Write these CER/WER values down in `README.txt`.

> **Note:** The test WER obtained using constrained filtering may be worse than vanilla greedy decoding. When the base model is poor, adding more diversity to sampling during decoding could hurt performance.

(C) Beam Search Decoding **[10 points]**

Beam search decoding is a standard alternative to greedy decoding. Run the cells under Task 2.3, with pointing `model` to the finetuned Whisper model. What is the test CER and WER you get with setting `beam_size` to 3? Write these CER/WER values down in `README.txt`. [**1 point**]

Constrained beam search. Modify the beam search decoding routine such that the output prediction is guaranteed to have **at least one English token**. We leave this as a broad requirement. Write down pseudocode for your algorithm, including all the assumptions you make, in the file `constrainedbeamsearch.pdf`. You will need to submit this file, along with your actual implementation in the notebook. Search for "TODO:" in the code cells under Task 2.4 in the Colab notebook. You can refer to `https://github.com/openai/whisper/blob/main/whisper/decoding.py#L301-L404` for the original implementation of the class `BeamSearchDecoder`, specifically its functions `update` and `finalize`. [**3 + 6 points**]

(D) Climb the Leaderboard **[5 points]**

This is an extra-credit part. You have free reign to improve further on the Hinglish ASR task while satisfying the following two constraints:

- Only available labeled training data you can use is in the 'train' split of the Hinglish corpus.
- You have to use the Whisper-small model.

You will output predictions for a blind test set that is listed in `test_blind.json` within `https://cse.iitb.ac.in/~pjyothi/cs753/dataset.zip`. Upload your predictions to the Kaggle task for this part to see where you appear on a leaderboard. Up to five extra credit points will be given to teams on the top of the leaderboard. If you are attempting this part, also submit a new notebook titled `assgmt2-extracredit.ipynb`.

Some ideas you could explore:

- Whisper supports prompting (e.g., see `https://arxiv.org/abs/2305.11095`). Could this be leveraged during finetuning to improve ASR performance?
- Extend constrained beam search to output at least one English word (rather than one English token).
- **Cheat resource.** Here is a 5K-sized list of all the words that appear in the dataset (including some distractors): `https://drive.google.com/file/d/1YfKK1INpS33ahqdm0nM3S3CGgzjnYKA6/view`. This also contains words appearing in the blind test set! Modify beam search decoding to make sure the predicted words are contained in this list.