CS 753 Spring 2022: Assignment #1

Instructor: Preethi Jyothi   Email: `pjyothi@cse.iitb.ac.in`   Total: 40 points

Februrary 7, 2022

Instructions: This assignment is due on or before 11.59 pm on February 28th, 2022. No extensions or grace period will be granted for this assignment.

- This is a group assignment. There are two parts to this assignment. In part 1, you will solve a language modeling task using WFSTs and the OpenFST toolkit. In part 2, you will build an ASR system for Wolof using the **Kaldi** toolkit.
- You will need to install the latest version of Kaldi available here. You can also install Kaldi using docker. (Instructions for using docker are available here.) **Please install the toolkit early and reach out to the TAs in case of any issues.**
- Click here for a detailed structure of your final submission directory. Parts of this assignment will be auto-graded. Hence, it is **very important that you do not deviate from the specified structure.** Deviations from the specified structure will be penalized. All the files that need to be submitted are highlighed in red below; all the submitted files will be within the parent directory `submission/`. Compress your submission directory using the command: `tar -cvzf submission.tgz submission` and upload `submission.tgz` to Moodle.

1. Is this Sentence Correct?

For this part of the assignment, you will need to use binaries from the OpenFST toolkit. If Kaldi is installed in the directory `[kaldi]`, then OpenFST binaries will be available within `[kaldi]/tools/openfst/bin`. (You can also install OpenFST from scratch by following the instructions here.)

(A) Fill in the blank **[10 points]**

A fixed vocabulary V of the 5000 most frequent words in English is listed in `vocab.txt`. Given an input sentence with a blank, your task is to fill in the blank with the most appropriate word from V using WFSTs.

- First, create an n-gram language model (LM) WFSA, L, with its input alphabet coming from `vocab.txt`. To create L:
  1. Scrape text from any publicly available English corpus with sentences containing words in V (e.g., Google's 1B word LM benchmark).
  2. Create a smoothed n-gram LM in the ARPA format. You can use any LM library that supports smoothed ngram LMs (e.g., KenLM, SRILM) to create an LM file in the ARPA format.
  3. Convert the ARPA format LM into an FST using the binary `[kaldi]/src/lmbin/arpa2fst` (that is part of the Kaldi toolkit).
- Next, create an FST F that encodes the input sentence with the blank. Finding the shortest path within the composed FST $F \circ L$ will give you the most appropriate word to fill in the blank.

You will need to submit two scripts, `1A/create-LM.sh` and `1A/fill.sh`.

`1A/create-LM.sh` will take vocab.txt as its input and create the FST `1A/L.fst`. An example run would look like:

Command Line

```
$ bash create-LM.sh vocab.txt
```

You should also submit the ARPA file as `1A/L.arpa`. The script `1A/create-LM.sh` will convert the ARPA format LM into an FST. (Please note you do not need to submit the text file that was used to create the ARPA LM. You can also assume that the required OpenFST utilities and `arpa2fst` will work with `1A` as the working directory. The TAs will set the PATH variable appropriately while grading.)

`1A/fill.sh` accepts a sentence as its input with the blank written as XXX and prints the best replacement for XXX to the standard output. (If necessary, we will enclose the input word in double quotes. E.g., "they're".) An example run of this script would look like:

Command Line

```
$ bash fill.sh the sun rose in the XXX
$ east
```

`1A/fill.sh` will internally call your Python program (and other shell scripts, if needed) to construct the FST `F.fst` and compose with `L.fst` (created via `create-LM.sh`) to give the required output. Include these scripts within the directory `1A/`, so that the call to `fill.sh` is successful from within `1A/`. (The Python script should run using Python 3.7 or Python 3.8.)

(B) Skeleton sentence **[8 points]**

Consider the following task.

Input: A tuple of two words (w_1, w_2), a set of two additional words $\{w_3, w_4\}$, and an LM L. All four words will be distinct.

Output: A sentence s with maximum probability according to L such that

- the words w_1 and w_2 should occur in s in that order (possibly with other intervening words), and
- the words w_3, w_4 should also occur in s (anywhere and in any order).

For this problem, you should use the same LM FST `1A/L.fst` that you created in the previous part 1A.

You should submit a script `1B/create-sent.sh` that accepts the four (space-separated) words w_1, w_2, w_3, w_4 from standard input and produces the output sentence on standard output with space-separated words. This script should internally create an FST T that encodes the desired constraints on the output sentence, and obtains the output as a shortest path in $T \circ L$ (ties can be broken arbitrarily). The correctness of your script will be evaluated as per the LM L that you submit for part A (provided that it is not a pathologically wrong L). Here are two sample runs of `create-sent.sh` with its sample outputs:

Command Line

```
$ bash create-sent.sh new york visit when
$ when should i visit new york
$ bash create-sent.sh you me tell time
$ can you tell me the time
```

2. Automatically Recognizing Speech in Wolof

Wolof is the most widely spoken language in Senegal and is written using the Latin alphabet. For this part of the assignment, you will develop an ASR system for Wolof. Download the baseline recipe at this link. If Kaldi is installed in the directory `[kaldi]`, untar `assgmt1.tgz` within `[kaldi]/egs` to get a directory `[kaldi]/egs/wolof`.

Note Set your working directory to be `[kaldi]/egs/wolof`. All subsequent evaluations will be done on evaluations sets in `data/dev` and `data/test`.

(A) Setting up the baseline system [3 points]

`run.sh` within `wolof` is the main wrapper script which you will be able to run in roughly 20 minutes at the end of this task. Go through this script carefully to understand the various steps involved. You can set the variable `stage` to determine which stages of `run.sh` will be processed. Also, note the messages on the command line when you execute `run.sh`. `decode.sh` is the most time-consuming of all the steps.

From your working directory, run the script `run.sh`:

Command Line

```
$ bash run.sh
```

This command will execute a number of scripts that prepare the datasets, create dictionary/LM FSTs, trains monophone HMMs and decodes a dev set to produce a word error rate (WER) of:

Command Line

```
%WER 58.79 [ 2191 / 3727, 109 ins, 398 del, 1684 sub ] exp/mono/decode_dev/wer_7
```

(You may get a WER that is close to 58.79% but not exactly this number due to hardware differences.) The following command within `run.sh` is used to train monophone HMMs for the acoustic model:

Command Line

```
        $ steps/train_mono.sh --nj 4 --cmd "$train_cmd" \
                data/train lang exp/mono
```

Within `steps/train_mono.sh`, look at the variables listed on lines 11–30 (within comments "Begin configuration section" and "End configuration section"). Figure out which variables are important by tuning on your dev set. Submit a text file `2A/wer.txt` that only contains the best WER on `data/test` you obtained using your final tuned hyperparameters. Also, submit a new `2A/train_mono.sh` with updated hyperparameters that we will use to train monophone HMMs and recover your reported number in `2A/wer.txt`. Note you are asked to report numbers on the test set (in `data/test`) by tuning your hyperparameters on the dev set (in `data/dev`).

Submit the file `run.sh` with relevant code for the four tasks mentioned below.

(B) Train tied-state triphone HMMs [3 points]

Uncomment the following lines to train tied-state triphone HMMs.

Command Line
```
$ steps/align_si.sh --nj "$nj" --cmd "$train_cmd" \
   data/train data/lang exp/mono exp/mono_ali
$ steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" \
   2000 20000 data/train data/lang exp/mono_ali exp/tri1
```

`steps/train_deltas.sh` takes two arguments 2000 and 20000. Figure out what these hyperparameters refer to. Tune them and observe how this tuning affects performance on the dev set. Submit the final, tuned hyperparameter values within `run.sh` in the call to `steps/train_deltas.sh`. (If you have edited `steps/train_deltas.sh`, please submit `2B/train_deltas.sh` that we will use to train tied-state triphone HMMs.) Also submit a text file `2B/wer.txt` that only contains the best WER you obtained on `data/test` using your final tuned hyperparameters that we will aim to reproduce.

(C) Augmentation **[4 points]**

Command Line
```
$ utils/data/perturb_data_dir_speed_3way.sh data/train data/train_sp3
```

Explore the effect of data augmentation. This can be implemented with the help of speed perturbations in `utils/data/perturb_data_dir_speed_3way.sh` (mentioned in the command above). Include a new stage in `run.sh` within an if clause `if [ $stage -le 5 ]` to implement this data augmentation step. Reestimate tied-state triphone models using the augmented data and decode the newly estimated models within this new stage. Submit a text file `2C/wer.txt` with your best WER on `data/test` that we will aim to reproduce by running the new stage 5 in `run.sh`.

(D) Concatenative Synthesis **[8 points]**

In this task, you will create a rudimentary text-to-speech system. Given a sequence of words w, you are required to find audio snippets for each word from the training utterances in `data/train` and concatenate them to create a new output speech file corresponding to w. (Make sure that all the speech snippets for the words in w come from the same speaker.) Submit a script `2D/basic-tts.sh` that when run as `./basic-TTS.sh "armeeli katolig yi"` will output an audio file `out.wav` containing the synthesized speech. You can use the commandline tool sox to snip an audio file. Any additional files or scripts you require should be within 2D. (You can assume that all the words in the input sentence will be found at least once in the training data.)
(Hint: You will need to force-align the training utterances to its transcripts in order to find timestamps where the utterance can be snipped.)

(E) Performance on blind test set **[4 points]**

This is the true test: How well does your Wolof ASR system perform on unseen utterances?

We will create a new directory `data/truetest/wav.scp` containing a sample set of unseen utterances. For your runs, you can populate this with test utterances from `data/dev` and/or `data/test`. We will replace this with a completely new set of unseen utterances during the blind test. Create a new stage in `run.sh` within `if [ $stage -le 6 ]`, with your best trained models and decode on `data/truetest`. Any files you need to run this new stage should be within 2E/. Any innovations are allowed in this new stage; e.g., check the scripts `steps/train_lda_mllt.sh` and `steps/train_sat.sh` that train speaker-adapted triphone HMMs. The only requirement is that you should stick to estimating HMM-based acoustic models. A leaderboard with the top-scoring N roll numbers and the corresponding WERs will be posted on Moodle. (N will depend on where there's a clean split in WERs.) You will receive full points for this question if your WER on the unseen utterances is lower than what we get using the baseline recipe. The N top-scoring performers on the leaderboard will gain extra credit points.

Useful Kaldi resources

- How to install Kaldi using docker
- Kaldi tutorial for beginners.
- Kaldi lecture slides.
- Kaldi troubleshooting.