Homework 2: GMM and Deep Acoustic Modeling

CS224S/LINGUIST285

Andrew Maas

Please read this entire page before beginning. And please do start early on this assignment.

1 Overview

Kaldi is a powerful ASR system developed in C++ that's used for speech recognition research here at Stanford to build state-of-the-art speech recognition systems, alongside many other techniques (which you'll learn more about in Andrew's lecture). Kaldi might also be a useful tool if you decide to build your final project on top of a well-written speech recognition system, but lack the time of writing it from scratch (as we always do). Detailed documentation/tutorial for Kaldi can be found here, which you might need to get familiar with if your final project involves Kaldi.

In this homework, you'll learn to

- Write Kaldi training scripts
- Improve ASR accuracy by tuning model parameters of the system
- Combine different techniques toward constructing a better ASR system
- Build a system that's used by the DMV, banks, and many other places to recognize digits

Next we'll walk you through the setup, then the task we would like you to explore, followed by the deliverables and finally a submission guide.

2 Setup

1. Log into Stanford FarmShare (corn) with your SUNet ID and password. (Students that are less familiar with SSH or FarmShare should feel free to drop by one of our office hours)

   ```
   ssh <Your_SUNet_ID>@corn.stanford.edu
   ```

2. We've provided you an script to setup the environment for this assignment, simply type

   ```
   /afs/ir/class/cs224s/hw_setup hw2
   ```

   in your command line window then press Enter.

3. Navigate to the directory where the running scripts of our assignment is

   ```
   cd hw2/tidigits/s5
   ```

4. Use your favorite text editor to view the contents of `run_mono.sh` (this is the first file you will be editing):

   ```
   vi run_mono.sh
   ```

5. Now, simply execute this script

   ```
   bash run_mono.sh
   ```

   Running the run_mono.sh script will typically take 2-3 minutes to finish. You will see many output from the script, most of them are Kaldi calling scripts checking the integrity of the data, or verifying the result of its data preparation scripts. By the time the script finishes, it will print two lines at the bottom, `WER 12.14 [ 3470 / 28583, 1060 ins, 379 del, 2031 sub ]` and `SER 24.18 [ 2104 / 8700 ]`, which indicate the word error rate and sentence error rate, respectively, of the baseline system that we've provided you.

   You might have noticed that our baseline recognizer did a very sloppy job recognizing digits – it missed every one out of ten digits! That means for a typical 16-digit credit card number, your phone banking system misses at least one number in it and makes you repeat 87% of the times.

3 Your Task

1. **Improve the monophone acoustic model**

   When you carefully scan `run_mono.sh`, you'll find that the script builds a simple monophone acoustic model over the training data. That is, to recognize each phone, it only considers the associated data but no contextual information (the preceding or succeding phones). Now, notice that the script employs `steps/train_mono.sh` to do the monophone training. This model is not tuned at all and has some deficiencies.

   In this part, your task is to modify the monophone acoustic model to improve the system's performance. Explore the parameters in `steps/train_mono.sh` and use your best judgment to make modifications to these parameters. Note that you will be making changes on `steps/train_mono.sh`.

   Deliverables: Report the adjustments you've made, and justify your decisions with your ASR knowledge. Report the resulting WER and SER. What are your observations on deletions, insertions and substitutions as you change the parameters? For example how do these numbers change as you increase boost silence probability? Explain. (Comment on two more parameters)

2. **Improve feature extraction**

   In the above training of the monophone acoustic model, we used raw MFCC features extracted from the audio files. As you might have made progress in improving the system performance with parameter tuning, the features that were used might not be ideal.

   In practice, speech recognition systems deal with all sorts of environments, where environmental noise might contribute greatly to hurt the performance of such systems, such as microphone quality, other people talking in the background, or probably the least you'll expect, refrigerator noise.

   To cope with such noise, we usually perform feature normalization after feature extraction is done. In run_mono.sh, find the section that is designated for this task, and observe the following line:

   ```
   steps/compute_cmvn_stats.sh --fake data/$x exp/make_mfcc/$x $mfccdir || exit 1;
   ```

   In the starter script we added the `-fake` flag in the CMVN normalization step to essentially skip that step. Now remove that flag, report your observations and explain in your report about what happens.

   Deliverables: Report the resulting WER and SER. What do you notice about the number of deletions, insertions and substitutions? Explain. How practical is your most current system for bank cards (16 digits, assume)?

3. **Try different training data**

   In case you're asking if there are other ways to improve your system, here's a quick and easy way. You might have noticed, while reading the `run_mono.sh` script, that we've provided you with a way to switch among different training sets. By default, your system so far have been trained on a reduced version of the TIDIGITS training set, which consists of only 10 male speakers out of TIDIGITS' 112-speaker training set of men and women.

   Specifically, four training sets were provided to you in this assignment.

Number	Description
1 (Default)	10 speakers, male only
2	10 speakers, female only
3	10 speakers, 5 male and 5 female
4	112 speakers, 55 male and 57 female (Full TIDIGITS training set)

   You can switch the training set easily by running the `run_mono.sh` script with the training set number as the argument. For example, to train your system on the full TIDIGITS training set, run:

```
bash run_mono.sh 4
```

   Deliverables: Tune the parameters in the monophone training script and report your modifications for **each** training set above. What are your observations on the system performance (WER, SER, deletions, insertions, substitutions)? Explain your findings. Now, normalize the features for each training set. In which model do you see the most improvement? Explain.

4. **Triphone acoustic model**

   Monophone acoustic model has limited power. For each phone to be recognized, it made no use of the contextual information about the preceding and succeeding phones. We will be fixing that in this section with a triphone acoustic model.

   In this task, you will be working with `run_tri.sh`. Before you begin, make sure that you have run `run_mono.sh` on the full TIDIGITS corpus (option 4) with your tuned parameters. When you're ready, open `run_tri.sh` and find the following line:

```
steps/train_deltas.sh --cmd "$train_cmd" \
    10 100 data/train data/lang exp/mono/aligned exp/tri
```

   To understand the meanings of the two numbers in the argument list, look up `steps/train_deltas.sh`. Your task is to tune these numbers in `run_tri.sh`.

   Deliverables: Report the tuned parameters. What is the system performance and how does it compare to the earlier versions? Justify your observations. Why do we call `steps/align_si.sh` in the script?

> **Note**: If you are unfamiliar with the idea of decision-tree based triphone acoustic models: due to the large volume of all possible triphones we could have, it is a common practice to cluster them into a decision tree, treat the tree leaves as the "new set of phones" (called senones) and build an acoustic model for them with GMMs (this should sound similar to what you've finished with monophone models).

5. **Neural Network Acoustic Model**

   In this task, you will train a small neural network that can be compared to your triphone model without writing actual code! The neural network model you will be training is a feed-forward neural network with sliding window. You will need to tune this model on the entire tidigits corpus.

   First make sure that you use the tuned parameters from the entire tidigits corpus, that is, complete tasks 1-4. Then start tuning the parameters in `run_nn.sh`. You should start from a smaller network and build towards a larger network. Be aware of overfitting!

Deliverables: You should report on the changes of your hyperparameters, and how it has affected your system's performance (both on training and test data). What happens when you increase the number of hidden layers? Is your final model performing better or worse than triphone model? Why?

4 Deliverables

Briefly describe your methods, results, and justifications (with your ASR knowledge learned in class) for each task. We expect you to understand the corresponding course material before starting on this project, and will grade you based on your demonstration of that understanding. You will be submitting this file on Gradescope. **Please submit a screenshot of each run script in the appendix.**

5 Tips and FAQ

1. In case you've overwritten our starter code `steps/train_mono.sh` for good, here are our original settings you might need for the comparisons:

```
num_iters=10    # Number of iterations of training
max_iter_inc=8 # Last iter to increase #Gauss on.
totgauss=100 # Target #Gaussians.
boost_silence=1.0 # Factor by which to boost silence likelihoods in alignment
realign_iters="1 4 7 10"; # Iterations on which the frames are realigned
```

2. WER and SER?

   Answer: At the end of the training script, several lines should be printed, some containing WERs (Word Error Rates), and others SERs (Sentence Error Rates). The definition of WER can be found in the slides, and the definition of SER is the total number of sentences you got wrong (had errors in your transcript) over the total number of sentences. In our case each sentence is a speech file of spoken numbers in the test set.