Lecture 6: HMM Refinement

Lecturer: Mark Hasegawa-Johnson (jhasegaw@uiuc.edu)
TA: Sarah Borys (sborys@uiuc.edu)
Web Page: http://www.ifp.uiuc.edu/speech/courses/minicourse/

June 2, 2005

This lecture will consider several different methods that can be used to make an HMM-based speech recognizer more complicated. All of these modifications have been shown to reduce error rate on some tasks, but there is no guarantee that any of these will reduce the error rate on any particular task of interest to you; it's always best to test a modification on independent data before accepting it permanently.

1 Data Without Phonetic Transcription

In many speech recognition applications, phonetic transcriptions are unavailable for part or all of the data. If phonetic transcriptions are available for part of the data, it's best to start with that data, and to initialize Gaussian monophone models as in lecture 5. If not, you can start with flat-start models using HCompV.

In order to start using the data without phonetic transcription, you need to know only the sequence of words uttered in each file. Create an MLF containing the word sequence, e.g.,

```
#!MLF!#
"*/sample.lab"
SILENCE__
this
is
a
sample
MLF
SILENCE__
.
```

You will also need to create a dictionary, e.g.,

```
MLF eh m eh l eh f
SILENCE__ sil
a ey
a ax
is ih z
sample s ae m pcl p el
this dh ih s
```

Notice that there are two possible pronunciations of the word "a."

Now, use the HLEd program to expand the word transcriptions into phoneme transcriptions. Create an edit file called something like `expand.hled`, containing the single line

```
EX
```

Then type

```
HLEd -d dict/my.dict -l '*' -i mlf/phn.mlf ed/expand.hled mlf/wrd.mlf
```

The result of expansion will be an MLF with phonemes listed in sequence, e.g.,

```
#!MLF!#
"*/sample.lab"
sil
dh
ih
s
ih
z
ey
s
ae
m
pcl
p
el
...
```

Notice that `HLEd` chooses the first available pronunciation of the word "a," and ignores any other possible pronunciations.

Once you have created a phoneme MLF, you can use `HERest` to retrain the phoneme HMMs. Since `HERest` ignores time alignment of the phones, the command line is exactly the same as in lecture 5.

2 Alternate Pronunciations

Some words, like "a," have really distinct pronunciations. Sometimes the different pronunciations have different meanings ("reject" as a noun vs. verb), sometimes they are appropriate in different phonological contexts ("the bird" vs. "the apple"), sometimes they are just in free variation. Once you have used `HERest` to adjust your HMMs to the new data, you can use `HVite` to pick out, from the dictionary, the most appropriate pronunciation of each word. Do something like

```
HVite -A -T 1 -l '*' -o SW -C cfg/train.cfg -a -H mmf/phones \
    -i mlf/align.mlf -m -y lab \
    -I mlf/wrd.mlf -S foo.scp dict/my.dict lists/phonelist.txt
```

The `-I` option specifies the input MLF, containing word-level transcriptions; the `-i` option specifies an MLF to which `HVite` should write the maximum-likelihood transcription of the data. The `-a` option specifies that `HVite` should operate in "forced alignment" mode, meaning that, instead of performing unconstrained speech recognition, it should only consider phoneme transcriptions that match the word transcription.

The resulting file, `align.mlf`, will contain the phoneme transcription that maximizes the probability of the observed speech, subject to the constraint that the phoneme transcription must match the word string. For example, if the talker said /ax/ rather than /ey/ for the word "a," the resulting MLF might read

```
#!MLF!#
"*/sample.lab"
0 800000 sil
800000 1500000 dh
1500000 3000000 ih
3000000 4000000 s
4000000 5500000 ih
5500000 6500000 z
6500000 8500000 ey
8500000 10000000 s
10000000 14000000 ae
...
```

3 Prosody-Dependent Phone Models

Suppose that you have a small prosodically labeled database, and a large prosodically unlabeled database. Using the methods sketched in lecture 5, you can train prosodically labeled phone models using the prosodically labeled data. Then you can create a dictionary in which each word has four alternate pronunciations: phrase-final vs. nonfinal, pitch accented vs. unaccented.

```
sample s_ ae_ m_ pcl_ p_ el_b
sample s_a ae_a m_a pcl_ p_ el_b
sample s_ ae_ m_ pcl_ p_ el_f
sample s_a ae_a m_a pcl_ p_ el_f
```

where `_a` means that a phoneme carries phrasal prominence (pitch accent), `_b` means that the phoneme is word-final (in the coda of the last syllable of the word), and `_f` means that the phoneme is phrase-final (in the coda of the last syllable of an intonational phrase).

If the prosodically labeled data and the unlabeled data are recorded under very similar recording conditions, it is possible to use `HVite` to choose among the different prosodic alternatives:

```
HVite -A -T 1 -l '*' -o SW -C cfg/train.cfg -a -H mmf/pd_phones \
    -i mlf/pd_align.mlf -m -y lab \
    -I mlf/wrd.mlf -S foo.scp dict/prosodic.dict lists/pd_phones.txt
```

where `prosodic.dict` is the dictionary shown above, `pros_dep_phones.txt` is a list of its phones, and the HMM file `mmf/pd_phones` must contain a trained model for each prosody-dependent phone. The resulting output file, `pd_align.mlf`, will contain the most probable pronunciation of each word—thus specifying whether the word is acoustically most similar to a phrase-final or non-final word, and whether it is most similar to a pitch-accented vs. unaccented word.

If the two databases are recorded under different conditions, `HVite` may be unable to find any sequence of HMMs that matches the data with high probability (the resulting HTK error is "No Tokens Survived Until End of File"). Here are some things that can help: (1) Generate all features and models using cepstral mean subtraction (CMS). CMS is implemented in HTK using the `_Z` modifier in all configuration and MMF files, e.g., `PLP_E_D_A_Z`. (2) When training HMMs on the unmatched data, require that the Gaussian PDF should never be allowed to develop a narrow variance. Minimum variance requirements can be specified using the `-v` option, e.g., `-v 0.2` (reasonable values of the variance can be guessed by examining the spectral data samples; use `HList`).

I have run into cases where a particular modified phoneme model is required by the prosody-dependent dictionary, but no examples are available in the prosodically labeled training data. For example, `aa_fa` may not exist in the prosodically labeled training data, but perhaps the non-final unaccented vowel `aa_` exists. This situation can be solved by (1) appending a "dummy phoneme" to the master model file, (2) using the HMM editing program `HHEd` to specify that the new phoneme is just a copy of the old one. A dummy phoneme can be appended to the file `pd_phones` by just changing the name in your phone prototype. `>>` is used to concatenate standard output of the `sed` command onto the end of text file `pd_phones`:

```
sed 's/proto/aa_fa/' mmf/proto >> mmf/pd_phones;
```

Then create an edit file, called something like `ed/merge.hhed`, with a line specifying that models `aa_fa` and `aa_` should both be TIed to the same underlying "macro," a macro called `aa_`:

```
TI aa_ {aa_fa,aa_}
```

Finally, call `HHEd` to create the new tied-state output file `mmf/pd_tied`:

```
HHEd -A -T 1 -H mmf/pd_phones -w mmf/pd_tied ed/merge.hhed lists/pd_phones.txt
```

Read through the file `mmf/pd_phones`, to make sure that the newly created macro contains the mean vectors and variances that were trained for the model `aa_`, instead of containing the "dummy" initialization values that you used for model `aa_fa`. Does it look OK? Then you should be ready to call `HVite` in order to align the prosody-dependent phones with the unlabeled training data.

4 Gender-Dependent Phone Models

In some cases, you will have a small amount of training data for which you know the gender of the speaker, and a large amount of data for which the gender of the speaker is unknown.

It is possible to train gender-dependent phone models using the gender-labeled data, and then to combine them into the same master model file. When they are combined into the same MMF, you will need to use some annotation in order to distinguish models designed for male talkers from those designed for female talkers. I use `_m` for male phonemes, and `_w` for female phonemes. Now each basic phoneme may have up to twelve different allophones. The content of the MMF might include:

```
~o
<STREAMINFO> 1 39
<VECSIZE> 39<NULLD><PLP_E_D_A><DIAGC>
~h "aa_m" ...
~h "aa_ma" ...
~h "aa_mb" ...
~h "aa_mba" ...
~h "aa_mf" ...
~h "aa_mfa" ...
~h "aa_w" ...
~h "aa_wa" ...
~h "aa_wb" ...
~h "aa_wba" ...
~h "aa_wf" ...
~h "aa_wfa" ...
```

The dictionary needs to be modified, in order to make use of the new phone set. Few people are able to change their gender in the middle of a word, therefore it's safe to assume that each word is either all male or all female:

```
simulate s_m ih_ma m_m y_m ax_m l_m ey_mba tcl_mb sp
simulate s_m ih_ma m_m y_m ax_m l_m ey_mfa tcl_mf sp
simulate s_m ih_m m_m y_m ax_m l_m ey_mb tcl_mb sp
simulate s_m ih_m m_m y_m ax_m l_m ey_mf tcl_mf sp
simulate s_w ih_wa m_w y_w ax_w l_w ey_wba tcl_wb sp
simulate s_w ih_wa m_w y_w ax_w l_w ey_wfa tcl_wf sp
simulate s_w ih_w m_w y_w ax_w l_w ey_wb tcl_wb sp
simulate s_w ih_w m_w y_w ax_w l_w ey_wf tcl_wf sp
```

I currently choose either the male or female version of each word using forced alignment. Obviously a talker should not change gender from one word to the next. If I know that the same talker is speaking for a given period of time, therefore, I post-process the `HVite` output: if there are more male phones associated with a given talker, then the talker is considered male, else female.

Gender-dependent recognition is perhaps better performed using a two-step strategy. In the first step, acoustic features that provide the best information about talker gender are observed (e.g., average F0), and are used to determine the gender of the talker. In the second stage, HMMs appropriate to the talker's gender are used.

5 Triphones

Phonemes change depending on phonetic context. The strongest context effects, by far, are caused by co-articulation of a phoneme with its right and left neighbors: e.g., the /k/ in "picket" is much different from the /k/ in "Bacall." Coarticulatory effects are modeled, in speech recognition, by splitting each phoneme into triphones. For example, the /k/ in "picket" would be the triphone `ih-k+ax` (/k/ as it occurs when preceded by /ih/ and followed by /ax/), while the /k/ in "Bacall" would be the triphone `ah-k+aa`. Although these two allophones are related, they are not acoustically identical.

Phonotactic constraints limit the number of triphones that can occur within a word in English to about 7000. Across word boundaries, the number of triphones that can occur is very nearly N^3, where N is the number of monophones. Fortunately, it is not really very important to model cross-word triphones: coarticulation effects across word boundaries are much weaker than within-word coarticulation. In order to block cross-word triphones in HTK, it is necessary to insert, into the dictionary and the MLF, a word boundary marker called "short pause" or sp. The dictionary is:

```
simulate s_m ih_ma m_m y_m ax_m l_m ey_mba tcl_mb sp
simulate s_m ih_ma m_m y_m ax_m l_m ey_mfa tcl_mf sp
simulate s_m ih_m m_m y_m ax_m l_m ey_mb tcl_mb sp
simulate s_m ih_m m_m y_m ax_m l_m ey_mf tcl_mf sp
simulate s_w ih_wa m_w y_w ax_w l_w ey_wba tcl_wb sp
simulate s_w ih_wa m_w y_w ax_w l_w ey_wfa tcl_wf sp
simulate s_w ih_w m_w y_w ax_w l_w ey_wb tcl_wb sp
simulate s_w ih_w m_w y_w ax_w l_w ey_wf tcl_wf sp
```

and the MLF is created by forced alignment. The short-pause model should only have one emitting state (three states total), and, as long as you never have two sp models in sequence, it is possible to create the transition matrix so that the recognizer can choose to skip over sp without ever outputting a spectrum:

```
~h "sp"
<BeginHMM>
 <NumStates> 3
 <State> 2
       <Mean> 39
        0.0 0.0 ...
       <Variance> 39
        1.0 1.0 ...
 <TransP> 3
 0.0 0.7 0.3
 0.0 0.6 0.4
 0.0 0.0 0.0
<EndHMM>
```

Once you have inserted the sp model in your dictionary, your MLF, and your MMF, it is possible to create triphones. Triphones are created by editing the MLF (in order to replace each monophone with the appropriate triphone), and by editing the MMF. The MLF is edited using `HLEd`, e.g.,

```
HLEd -n lists/triphones.txt -l '*' -i mlf/triphones.mlf ed/triphones.hled mlf/gabm.mlf;
```

where `-n` asks for a list of all of the triphones used in the output MLF, `triphones.mlf`, and the edit file `triphones.hled` contains `WB` commands specifying the word boundary symbols, and a `TC` command that performs the triphone expansion:

```
WB sp
WB sil
TC
```

Editing the MMF is done using `HHEd`. The `HHEd` command `CL lists/triphones.txt`, in a file such as `triphones.hhed`, causes `HHEd` to CLone each monophone model as many times as necessary in order to initialize all of the triphones requested by the file `lists/triphones.txt`. `HHEd` only creates the requested triphones, because as mentioned above, the number of requested triphones is likely to be many orders of magnitude less than the total number of all possible triphones. Given an edit file containing the CL command, the following command line will create triphones, where `pd_phones.txt` contains a list of the non-triphones (i.e., all the phones in `triphones.txt` must be built from the phones in `pd_phones.txt`):

```
HHEd -B -A -T 1 -w mmf/triphones -H mmf/pd_phones ed/triphones.hhed lists/pd_phones.txt
```

The `-B` option tells `HHEd` to store the output in a compact binary format instead of a text format.

6 Clustered Triphones

You don't really want to train 7000 triphone models using any database with fewer than several hundred hours of speech. It is better to cluster the triphones. "Clustered triphones" are triphones that depend not on the phoneme labels of both neighbors, but instead, only on the class of the neighboring phones; for example, given that the neighboring phone is a vowel, /k/ might only be sensitive to whether it is a front vowel or a back vowel.

HTK is distributed with a useful tree-based clustering algorithm [1]. The tree-based clustering algorithm accepts a long list of allowable phonetic class distinctions, or "question," phrased in the following form:

```
QS "L_Nasal" { ng-*,n-*,m-* }
QS "R_Nasal" { *+ng,*+n,*+m }
```

The first "question" specifies that if the left phone is /m,n,ng/, then it is nasal, otherwise not. The second "question" specifies a similar distinction for the right phone. Obviously, there are an enormous number of possible phonetic distinctions that one might ask about. The `HHEd` command `TB` examines the statistics of the training corpus, in order to determine which of these possible questions is most useful for each phoneme.

The first step in tree-based clustering is to generate a statistics file, by running `HERest` once. The following command will accumulate the statistics necessary for tree-based clustering, and save them in `mmf/triphone.stats`. It will also update the models, and save them in `mmf/t2/triphones`, but if you really don't think that you have enough data to accurately estimate the triphones, then you can just ignore the re-estimated MMF.

```
HERest -A -s mmf/triphone.stats -I mlf/triphones.mlf -B \
 -C cfg/train.cfg -S scp/train.scp -H mmf/triphones \
-M mmf/t2 lists/triphones.txt;
```

The second step is to create the `HHEd` edit file, called, for example, `cluster.hhed`. `cluster.hhed` will be much larger than any other edit file that you've ever seen. The first line reads in the statistics:

```
RO 20.0 "mmf/k01i1/triphone.stats
```

The next many, many lines specify the questions. A list of questions is a valuable resource, that should be saved and re-used from one recognizer to the next:

```
QS "L_Fricative" { s-*,f-*,th-*,sh-*,z-*,v-*,dh-*,zh-* }
...
QS "R_Palatovelar" { *-sh,*-zh,*-y,*-k,*-g }
```

The next many questions specify the particular states that you want to consider clustering. Each `TB` command specifies the number of tokens per output leaf, the base name of the macros that will be tied together by this command, and the list of states or HMMs that may be tied together. You may want to generate the TB commands automatically from your phone list, using perl or python. It is possible to tie together either the whole HMM, or the individual states. The following command, for example, specifies that HTK should consider how best to cluster the second states of all of the different triphones based on the gender-dependent phone `aa_m`:

```
TB 100.0 "aa_mS2" {(aa_m,*-aa_m,aa_m+*,*-aa_m+*).state[2]}
```

The `TB` commands grow the trees, but the `AU` command actually creates the new clustered-triphone acoustic models, by merging models from the input triphone list `lists/triphones.txt`. Finally, the `ST` command is used to write out the trees, and the `CO` command is used to write out a list of the new clustered-triphone models:

```
ST lists/triphone_trees.txt
CO lists/clustered.txt
```

The third step is to actually run `HHEd` to implement the commands in your edit file, and to write out the resulting models to the file `mmf/clustered`:

```
HHEd -A -B -T 1 -H mmf/triphones -w mmf/clustered ed/cluster.hhed lists/triphones.txt
```

7 Mixture Gaussians

Triphone clustering can only be done using Gaussian PDF, not mixture Gaussian (the distance metric used to decide which triphones should be clustered is only meaningful for a Gaussian PDF). For that reason, we stick to Gaussian PDFs until after we are done creating clustered triphones.

Once you have created clustered triphones, it may be useful to increase the precision of the observation PDF by "upmixing" from Gaussian to mixture Gaussian PDFs.

Upmixing is performed iteratively. First, use several iterations of HERest to re-estimate the Gaussian-PDF clustered triphone models. Then upmix to two or three mixtures per state. Then re-estimate the 3-mixture models before upmixing to 5 mixtures per state, and so on. For example

```
HERest -A -I mlf/triphones.mlf -C cfg/train.cfg -S scp/train.scp \
  -H mmf/clustered -M mmf/c2 lists/clustered.txt;
HERest -A -I mlf/triphones.mlf -C cfg/train.cfg -S scp/train.scp \
  -H mmf/c2/clustered -M mmf/c3 lists/clustered.txt;
HHEd -A -T 1 -H mmf/c3/clustered -M mmf/3mix ed/upmix03.hhed \
  lists/clustered.txt;
HERest -A -I mlf/triphones.mlf -C cfg/train.cfg -S scp/train.scp \
  -H mmf/3mix/clustered -M mmf/3mix/c2 lists/clustered.txt;
HERest -A -I mlf/triphones.mlf -C cfg/train.cfg -S scp/train.scp \
  -H mmf/3mix/c2/clustered -M mmf/3mix/c3 lists/clustered.txt;
HHEd -A -T 1 -H mmf/3mix/c3/clustered -M mmf/5mix ed/upmix05.hhed \
  lists/clustered.txt;
...
```

and so on. The files `upmix03.hhed` and `upmix05.hhed` contain just one line each:

```
MU 03 { *.state[2-4].mix }
```

or

```
MU 05 { *.state[2-4].mix }
```

As you add mixtures, training corpus accuracy of the recognizer will continuously improve, but test corpus accuracy will eventually reach a maximum before starting to decline. It is wise to find the optimum number of mixtures using cross-validation: after applying `HERest` to each upmixed model set, test it on a held-out development test set to see how well it works. When performance on the devtest set stops improving, stop adding mixtures.

References

[1] J. J. Odell, P. C. Woodland, and S. J. Young. Tree-based state clustering for large vocabulary speech recognition. In *Proc. Internat. Sympos. Speech, Image Process. and Neural Networks*, pages 690–693, Hong Kong, 1994.