Lecture 5: Hidden Markov Models

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

May 27, 2005

1 Training Monophone Models Using HTK

1.1 Installation

Download HTK from http://htk.eng.cam.ac.uk/. You should download the standard distribution (gzipped tar file). You may also wish to download the samples.tar.gz file, which contains a demo you can run to test your installation. You may also wish to download the pre-compiled PDF copy of the HTKBook.

Compile HTK as specified in the README file. Under windows, you will need to use a DOS prompt to compile, because the VCVARS32.bat file will not run under cygwin.

Add the bin.win32 or bin.linux directory to your path. In order to test your distribution, move to the samples/HTKDemo directory, and (assuming you are in a cygwin window by now) type ./runDemo.

1.2 Readings

1. Primary readings are from the HTKBook [3]. Before you begin, read sections 3.1.5-3.4 and 6.2. Before you start creating acoustic features, read sections 5.1-5.2, 5.4, 5.6, 5.8-5.11, and 5.16. Before you start training your HMMs, read sections 7.1-7.2 and 8.1-8.5.
2. Those who do not already know HMMs may wish to read either HTKBook chapter 1, or read Rabiner [2] and Juang et al. [1]. Even those who already know HMMs may be interested in the discussion of HTK's token-passing algorithm in section 1.6 of the HTKBook.

1.3 Creating Label and Script Files

A *script* file in HTK is a list of speech or feature files to be processed. HTK's feature conversion program, HCopy, expects an ordered list of pairs of input and output files. HTK's training and test programs, including HCompV, HInit, HRest, HERest, and HVIte, all expect a single-column ordered list of acoustic feature files. For example, if the file TRAIN2.scp contains

```
d:/timit/TIMIT/TRAIN/DR8/MTCS0/SI1972.WAV data/MTCS0SI1972.MFC
d:/timit/TIMIT/TRAIN/DR8/MTCS0/SI2265.WAV data/MTCS0SI2265.MFC
...
```

then the command line "HCopy -S TRAIN2.scp . . . " will convert SI1972.WAV and put the result into data/MTCS0SI1972.MFC (assuming that the "data" directory already exists). Likewise, the command "HInit -S TRAIN1.scp . . . " works if TRAIN1.scp contains

```
data/MTCS0SI1972.MFC
data/MTCS0SI2265.MFC
...
```

The long names of files in the "data" directory are necessary because TIMIT files are not fully specified by the sentence number. The sentence SX3.PHN, for example, was uttered by talkers FAJW0, FMBG0, FPLS0, MILB0, MEGJ0, MBSB0, and MWRP0. If you concatenate talker name and sentence number, as shown above, the resulting filename is sufficient to uniquely specify the TIMIT sentence of interest.

A *master label file* (MLF) in HTK contains information about the order and possibly the time alignment of all training files or all test files. The MLF must start with the seven characters "#!MLF!#" followed by a newline. After the global header line comes the name of the first file, enclosed in double-quote characters ("); the filename should have extension .lab, and the path should be replaced by "*". The next several lines give the phonemes from the first file, and the first file entry ends with a period by itself on a line. For example:

```
#!MLF!#
"*/SI1972.lab"
0 1362500 sil
1362500 1950000 p
21479375 22500000 sil
.
"*/SI1823.lab"
...
```

In order to use the initialization programs HInit and HRest, the start time and end time of each phoneme must be specified in units of 100ns (10 million per second). In TIMIT, the start times and end times are specified in units of samples (16,000 per second), so the TIMIT PHN files need to be converted. The times shown above in 100ns increments, for example, correspond to the following sample times in file SI1972.PHN:

```
0 2180 h#
2180 3120 p
...
```

Notice that the "h#" symbol in SI1972.PHN has been changed into "sil". TIMIT phoneme labels are too specific; for example, it is impossible to distinguish "pau" (pause) from "h#" (sentence-initial silence) or from "tcl" (/t/ stop closure) on the basis of short-time acoustics alone. For this reason, when converting .PHN label files into entries in a MLF, you should also change phoneme labels as necessary in order to eliminate non-acoustic distinctions. Some possible label substitutions are s/pau/sil/ (silence), s/h#/sil/, s/tcl/sil/, s/pcl/sil/, s/kcl/sil/, s/bcl/vcl/ (voiced closure), s/dcl/vcl/, s/gcl/vcl/, s/ax-h/axh/, s/axr/er/, s/ix/ih/, s/ux/uw/, s/nx/n/ (nasal flap), s/hv/hh/. The segments /q/ (glottal stop) and /epi/ (epinthetic stop) can be deleted entirely.

All of the conversions described above can be done using a single perl script that searches through the TIMIT/TRAIN hierarchy. Every time it finds a file that matches the pattern `S[IX]\d+.PHN` (note: this means it should ignore files SA1.PHN and SA2.PHN), it should add necessary entries to the files TRAIN1.scp, TRAIN2.scp, and TRAIN.mlf, as shown above. When the program is done searching the TIMIT/TRAIN hierarchy, it should search TIMIT/TEST, creating the files TEST1.scp, TEST2.scp, and TEST.mlf. See the scripts `peruse_tree.pl` and `sph2mlf.pl` in `transcription_tools.tgz` for an example. Alternatively, you could create the scripts by doing the tree searching in bash, and use perl for just the regex substitution:

```
for d in ‘ls TRAIN‘; do
  for s in ‘ls TRAIN/$d/‘; do
   ls TRAIN/$d/$s/S[IX]*.WAV | ‘ |
     perl -e \
      ’while($a=<>){$b=$a;$a=~s/.*\//data\//;$a=~s/\.WAV/\.mfc/;print "$b  $a\n";}’ \
      >> TRAIN2.scp;
  done
done
```

Finally, just in case you are not sure what phoneme labels you wound up with after all of that conversion, you can use perl's hash-table feature to find out what phonemes you have:

```
awk ’/[\.!]/{next;}{print \$3}’ TRAIN.mlf | sort | uniq > monophones
```

The first block of awk code skips over any line containing a period or exclamation point. The second block of awk code looks at remaining lines, and prints out the third column of any such lines. The unix sort and uniq commands sort the resulting phoneme stream, and throw away duplicates.

1.4 Creating Acoustic Feature Files

Create a configuration file similar to the one in HTKBook page 32. Add the modifier `SOURCEFORMAT=NIST` in order to tell HTK that the TIMIT waveforms are in NIST format.

I also recommend a few changes to the output features, as follows. First, compute the real energy (MFCC_E) instead of the cepstral pseudo-energy (MFCC_0). Second, set `ENORMALISE` to `T` (or just delete the `ENORMALISE` entry).

Use `HCopy` to convert TIMIT waveform files into MFCC, as specified on page 33 of the HTK book. Convert both the TRAIN and TEST corpora of TIMIT.

1.5 HMM Training

Use a text editor to create a prototype HMM with three emitting states (five states total), and with one Gaussian PDF per emitting state (see Fig. 7.3). Be sure that your mean and variance vectors contain the right number of acoustic features: three times the number of cepstral coefficients, plus three energy coefficients.

Change your configuration file: eliminate the `SOURCEFORMAT` specifier, and change `TARGETKIND` to `MFCC_E_D_A`.

Use `HCompV` as specified on page 34 to create the files hmm0/proto and hmm0/vFloors. Next, use your text editor to separate hmm/macros (as shown in Fig. 3.7) from the rest of the file hmm0/proto (the first line of hmm0/proto should now read `~h "proto"`).

Because your .lab files specify the start and end times of each phoneme in TIMIT, you can use HInit and HRest to initialize your HMMs before running HERest. Generally, the better you initialize an HMM, the better it will perform, so it is often a good idea to use HInit and HRest if you have relevant labeled training data. Run HInit as shown on page 120, using `$phn` as your variable, e.g.,

```
#!/bin/bash
mkdir hmm1;
for phn in `cat monophones`; do
  HInit -I TRAIN.mlf -S TRAIN1.scp -H hmm0/macros \
      -C config -T 1 -M hmm1 -l $phn hmm0/proto;
  sed "s/proto/$phn/" hmm1/proto > hmm1/$phn;
done
```

Hint: once you have the lines above working for one phoneme label, put them inside a for loop to do the other phonemes.

Re-estimate the phonemes using HRest, as shown on page 123. Again, once you have the function working for one phoneme, put it inside a for loop. HRest will iterate until the log likelihood converges (use the -T 1 option if you want to see a running tally of the log likelihood), or until it has attempted 20 training iterations in a row without convergence. If you want to allow HRest to iterate more than 20 times per phoneme (and if you have enough time), specify the -i option (I used -i 100).

Once you have used HRest, you may wish to combine all of the trained phoneme files into a single master macro file (MMF). Assuming that all of your phoneme filenames are 1-3 characters in length, and that the newest versions are in the directory hmm2, they can be combined by typing

```
cat hmm2/? hmm2/?? hmm2/??? > hmm2/hmmdefs
```

Now run the embedded re-estimation function HERest to update all of the phoneme files at once. HERest improves on HRest because it allows for the possibility that transcribed phoneme boundaries may not be precisely correct. HERest can also be used to train a recognizer even if the start and end times of individual phonemes are not known.

Unfortunately, HERest only performs one training iteration each time the program is called, so it is wise to run HERest several times in a row. Try running it ten times in a row (moving from directory hmm2 to hmm3, then hmm3 to hmm4, and so on up to hmm12). Hint: put this inside a for loop.

1.6 Testing

In order to use HVIte and HResults to test your recognizer, you first need to create a "dictionary" and a "grammar."

For now, the "grammar" can just specify that a sentence may contain any number of phonemes:

```
$phone = aa | ae | ... | zh ;
( <$phone> )
```

Parse your grammar using HParse as specified on page 27.

The "dictionary" essentially specifies that each phoneme equals itself:

```
aa aa
ae ae
...
```

Because the dictionary is so simple, you don't need to parse it using HDMan. You can ignore all of the text associated with Fig. 3.3 in the book.

Run HVIte as specified in section 3.4.1 of the book; instead of "tiedlist," you should use your own list of phonemes (perhaps you called it "monophones"). You may have to specify -C config, so that HVite knows to compute delta-cepstra and accelerations. The -p option specifies the bonus that HVIte gives itself each time it inserts a new word. Start with a value of -p 0. Use -T 1 to force HVIte to show you the words it is recognizing as it recognizes them. If there are too many deletions, increase -p; if there are too many insertions, decrease -p.

When you are done, use HResults to analyze the results:

```
HResults -I TEST.mlf monophones recout.mlf
```

You should get roughly 55-60% correct, and your recognition accuracy should be somewhere in the range 40-60%. These terms are defined as follows:

$$\text{CORRECTNESS} = 100 \times \frac{\text{NREF} - \text{SUBSTITUTIONS} - \text{DELETIONS}}{\text{NREF}}$$

$$\text{ACCURACY} = 100 \times \frac{\text{NREF} - \text{SUBSTITUTIONS} - \text{DELETIONS} - \text{INSERTIONS}}{\text{NREF}}$$

Correctness is equal to the percentage of the reference labels (NREF) that were correctly recognized. Correctness does not penalize for insertion errors. Accuracy is a more comprehensive measure of recognizer quality, but it has many counter-intuitive properties: for example, Accuracy is not always between 0 and 100 percent. Recent papers often use the terms Precision and Recall instead, where Recall is defined to equal Correctness, and Precision is the percentage of the recognized labels that are correct, i.e.,

$$\text{PRECISION} = 100 \times \frac{\text{NRECOGNIZED} - \text{SUBSTITUTIONS} - \text{INSERTIONS}}{\text{NRECOGNIZED}}$$

$$\text{NRECOGNIZED} = \text{NREF} - \text{DELETIONS} + \text{INSERTIONS}$$

2 Words and Triphones

In this section, you will use the TIMIT monophone HMMs trained in the previous lecture as the starting point for a clustered triphone recognizer designed to transcribe the words spoken by a talker in the BU Radio News corpus.

2.1 Readings

The HTK Book, chapters 10, 12, and sections from 14 about HBuild, HLStats, HHEd and HLEd.

2.2 Cepstral Mean Subtraction; Single-Pass Retraining

If $\vec{x}_t$ is a log-spectral vector or a cepstral vector, the frequency response of the microphone and the room will influence only the average value of $\vec{x}_t$. It is possible to reduce the dependence of your recognizer on any particular microphone by subtracting the average value of $\vec{x}_t$, averaged over an entire utterance, before training or testing the recognizer, i.e.,

$$\vec{y}_t = \vec{x}_t - \frac{1}{T}\sum_{t=1}^{T}\vec{x}_t \tag{1}$$

Equation 1 is called cepstral mean subtraction, or CMS. In HTK, CMS is implemented automatically if you append "_Z" to the feature specification. For example, you can save the features as type `MFCC_E`, then use a configuration file during training and testing that specifies a feature vector of type `MFCC_E_D_A_Z`.

HTK offers a method called "one-pass retraining" (HTKBook section 8.X) that uses models trained with one type of feature vector (for example, `MFCC_E_D_A`) in order to rapidly train models with a different feature vector type (for example, `MFCC_E_D_A_A`). In theory, you need to have available files of both data types, but since HTK can implement CMS on the fly when opening each feature file, there is no need to regenerate the training data. Just create a script file with two columns — the "old" and "new" feature files, which in this case are the same file:

```
data/SI1972.MFC data/SI1972.MFC
...
```

Then create a configuration file with entries `HPARM1` and `HPARM2`, as specified in section 8.X of the HTKBook, and call `HERest` with the `-r` option, exactly as specified in that section. Compare the hmmdefs files for the old and new file types. You should notice that the feature file type listed at the top of each file has changed. You should also notice that the mean vectors of each Gaussian mixture have changed a lot, but the variance vectors have not changed as much.

2.3 Dictionaries

In order to recognize words using sub-word recognition models, you need a pronunciation dictionary. Pronunciation dictionaries for talker F1A in the Radio News corpus are provided in F1A/RADIO/F1A.PRN and F1A/LABNEWS/F1ALAB.PRN.

These dictionaries contain a number of diacritics that will be useful later, but are not useful now. Use sed, awk, or perl to get rid of the characters * and — (syllable markers), and the notation "+1" or "+2" in any transcription line. In order to reduce the number of homonyms, you may also wish to convert all capital letters to lower-case (so that "Rob" and "rob") are not distinct), and also eliminate apostrophes (so that "judges" and "judges' " are not distinct). You will also wish to make a few label substitutions in order to map radio news phonemes into the TIMIT phonemes defined last week: axr becomes er, pau and h# become sil, and every stop consonant (b,d,g,p,t,k) gets split into two consecutive TIMIT-style phones: a closure followed by a stop release.

As an example, the radio news dictionaries might contain the following entry for "Birdbrain's"

```
Birdbrain's b axr+1 d * b r ey n z
```

Assuming that your TIMIT-based phoneme set includes er but not axr, you would wish to automatically translate this entry to read

```
birdbrains vcl b er vcl d vcl b r ey n z
```

or, by adding a rule that deletes the stop release when the following segment is another consonant, you might get

```
birdbrains vcl b er vcl vcl b r ey n z
```

Notice that there is another alternative: instead of modifying the dictionary to match your HMM definitions, you could modify your HMM definitions to match the dictionary. Specifically, er could be relabeled as axr, sil could be relabeled as h#, and you could concatenate your stop closure and stop release states in order to create new stop consonant models. You could even create models of '*' and '—' with no emitting states.

Once you have converted your dictionaries, you should concatenate them together, then apply the unix utilities sort and uniq to the result, e.g., `convert_dict.pl F1A/RADIO/F1A.PRN F1A/LABNEWS/F1ALAB.PRN | sort | uniq > F1A.dict`. HTK utilities will not work unless the words in the dictionary are orthographically sorted (alphabetic, all capitalized words before all diminutive words).

2.4 Transcriptions

Create master label files using almost the same perl script that you used for TIMIT, but with .WRD-file inputs instead of .PHN-file inputs. Also, every waveform file in RADIO NEWS is uniquely named, so you don't need to concatenate the directory and filename. The resulting master label files should look something like this, although the start times and end times are completely optional:

```
#!MLF!#
"*/F1AS01P1.lab"
0 2100000 a
2100000 4700000 cape
4700000 7600000 cod
```

In order to train the grammar, you need a word-level master label file, as shown above. In order to train the HMMs, though, you need a phoneme-level master label file. The phone-level MLF can be computed from the dictionary + word-level-MLF using the `HLEd` command (see section 12.8 in the HTK Book). Create a file called `expand.hled` that contains just one command,

```
EX
```

Then type

```
HLEd -d F1A.dict -l '*' -i phone\_level.mlf expand.hled word\_level.mlf
```

If `HLEd` fails, the most likely cause is that your master label file contains entries with times but no words. `HLEd` will, unfortunately, not tell you where those entries are. Try printing out all lines that have less than three columns using a command like

```
gawk 'nf<3{print}' word\_level.mlf
```

Scan the output to make sure that you don't have phantom "words" with start times and end times but no word labels.

2.5 Creation of MFCCs

Create a two-column and a one-column script file for your training data, and the same for your test data, just as you did for TIMIT. The two-column script file will look something like:

```
d:/radio_news/F1A/RADIO/S01/F1AS01P1.SPH    data/F1AS01P1.MFC
d:/radio_news/F1A/RADIO/S01/F1AS01P2.SPH    data/F1AS01P2.MFC
```

You may use any subset of the data for training, and any other subset for test. I trained speaker-dependent HMMs using the F1A/RADIO directory, and tested using the F1A/LABNEWS directory. You may get better recognition results if your training and test set both include part RADIO data and part LABNEWS data.

Use `HCopy` to convert waveforms to MFCCs.

2.6 Bigram Grammar

Construct a list of your entire vocabulary, including both training and test sets, using

```
awk '{print $1}' F1A.dict | sort | uniq > wordlist
```

where `$1` is awk's notation for the entry in the first text column. Seeding your grammar with words from the test set is cheating, but for datasets this small, it may be the only way to avoid huge numbers of out-of-vocabulary errors.

Given a master label file for your entire training data, the command HLStats will compute a backed-off bigram language model for you, and HBuild will convert the bigram file into a format that can be used by other HTK tools. See sections 12.4 and 12.5 in the HTKBook for examples; note that you will need to specify both the -I and -S options to HLStats.

2.7 Monophone HMMs

If your dictionary matches the labels on your TIMIT monophone models, you should be able to use the TIMIT models now to perform recognition on the radio news corpus. Try it:

```
HVIte -C config_recog -H timit/macros -H timit/hmmdefs -S (one-column test script) \
    -l '*' -i recout1.mlf -t 250.0 -w (HBuild output file) \
    -p 5 -s 3 F1A.dict monophones
HResults -I (test MLF) monophones recout1.mlf
```

The `-p` and `-s` options set the word insertion penalty and the grammar weight, respectively. These parameters are described in section 13.3 of the HTKBook. Adjusting these parameters can cause huge changes in your recognition performance; 5 might or might not be a good value.

In any case, your results will probably be pretty horrible. Radio news was recorded using different microphones than TIMIT, by different talkers. You can account for these differences by adapting the models (using `HEAdapt`) or by re-estimating them (using `HERest`) — you probably have enough data to use re-estimation instead of adaptation.

Re-estimate your models using `HERest`, and then run `HVIte` again. Your results should improve somewhat, but may still be disappointing. How can you improve your results still further?

2.8 Word-Internal Triphones

In order to use word-internal triphones, you need to augment your transcriptions using a special word-boundary "phoneme" label. The `sp` (short pause) phoneme is intended to represent zero or more frames of silence between words.

Add `sp` to the end of every entry in your dictionary using awk or perl. *After* you have added `sp` to the end of every entry, add another entry of the form

```
silence sil
```

The "silence" model must not end with an `sp`.

Now you need to augment your HMM definitions, exactly as listed in section 3.2.2 and 3.2.3 of the HTK book. This consists of four steps. First, add `sp` to the end of your `monophones` list file. Second, edit your hmmdefs file with a text editor, in order to create the `sp` model by copying the middle state of the `sil` model. Third, use `HHEd` with the script given in 3.2.2. Finally, use `HVIte` in forced-alignment mode, in order to create a new reference transcription of the training data. Be sure to use the `-b silence` option to add silences to the beginning and end of each transcription; otherwise your sentence will end with an `sp` model, and that will cause `HERest` to fail.

Now that you have your word-boundary marker, you are ready to create word-internal triphones. Use `HLEd` exactly as in section 3.3.1 of the HTKBook. Because of the small size of this database, the test set may contain triphones missing from the training data. In order to accomodate missing triphones, concatenate the monophone and triphone files, so that any missing triphones can at least be modeled using monophones:

```
sort monophones triphones | uniq > allphones
```

Finally, use `HHEd` as in section 3.3.1 of the HTKBook, but use the `allphones` list instead of the `triphones` list to specify your set of output phones.

Re-estimate your triphone models a few times using `HERest`. `HERest` will complain that some triphones are observed only one or two times in the training data. I guess we need a larger training database.

Test the result using `HVIte`. The presence of monophones in your phoneme list will confuse `HVIte`. In order to force the use of triphones whenever possible, your config file should contain the entries

```
FORCECXTEXP = T
ALLOWXWRDEXP = F
```

Your recognition performance with triphones should be better than it was with monophones.

2.9 Tied-State Triphones

Because of the sparsity of the training data, many triphone models are not well trained. The problem can be alleviated somewhat by using the same parameters in multiple recognition models. This process is called "parameter tying."

Chapter 10 of the HTKBook describes many, many different methods of parameter tying, all of which are frequently used in practical recognition systems. I suggest using the data-driven clustering method for the current exercise (section 10.4), although tree-based clustering (section 10.5) might work almost as well.

Run `HERest` with the `-s` option, in order to generate a file called `stats_file`. Then create an `HHEd` script that starts with the command `RO (threshold) stats_file` where `(threshold)` specifies the minimum expected number of times a state should be visited in order to count for parameter tying (I used 20).

Use perl, awk, or even just bash to add commands of the following form to your `HHEd` script:

```
TC 100.0 "aaS2" {(aa,*-aa,aa+*,*-aa+*).state[2]}
TC 100.0 "aaS3" {(aa,*-aa,aa+*,*-aa+*).state[3]}
TC 100.0 "aaS4" {(aa,*-aa,aa+*,*-aa+*).state[4]}
```

You can be more general, if you like. For example, the following command would allow HTK to consider tying together the first state of `aa` with the last state of any phoneme that precedes `aa`:

```
TC 100.0 "aaS2" {(aa,*-aa,aa+*,*-aa+*).state[2],(*-*+aa,*+aa).state[4]}
```

Run `HHEd` in order to perform data-based tying (use the `-T` option to see what `HHEd` is doing). Use `HERest` to re-estimate the models a few times, then test using `HVIte` and `HResults`. Your performance may still not be wonderful, but it should be better than you obtained without parameter tying.

References

[1] Bin H. Juang, Stephen E. Levinson, and Man Mohan Sondhi. Maximum likelihood estimation for multivariate mixture observations of Markov chains. *IEEE Trans. on Information Theory*, 32(2):307–309, 1986.

[2] Lawrence R. Rabiner and Bing-Hwang Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–16, January 1986.

[3] Steve Young, Gunnar Evermann, Thomas Hain, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book*. Cambridge University Engineering Department, Cambridge, UK, 2002.