Lecture 4: Overview of Automatic Speech Recognition

Lecturer: Mark Hasegawa-Johnson (jhasegaw@uiuc.edu)
TA: Sarah Borys (sborys@uiuc.edu)

January 10, 2009

Contents

1 The Motivation for All of the Design Decisions Made in Standard ASR Systems

An automatic speech recognizer is a program that listens to an audio file, and writes down the words that it contains (Fig. 1). The criterion for success is the word error rate (the number of mistakes in the automatic transcription) [9]. In other words, suppose that we want to design our system to minimize the expected number of word errors in an N-word sentence. Suppose that w_l, $1 \le l \le L$, are the correct words (the words the talker actually said), and $\hat{w}_l$ are the words transcribed by the ASR (we'll ignore insertion and deletion errors for now). The expected number of errors in the sentence is then:

$$E\left[\text{WER}\right] = \frac{1}{L}\sum_{l=1}^{L} P(w_l \neq \hat{w}_l | X) \tag{1}$$

where X is the audio file that we're supposed to be transcribing.

Suppose that the ASR program is known (already designed), and we want to choose a word $\hat{w}_l$ to minimize the expected error rate. The best word to choose, $\hat{w}_l^*$, is given by the maximum *a posteriori* (MAP) decision rule:

$$\begin{aligned}
\hat{w}_l^* &= \arg\min_{\hat{w}} E\left[\text{WER}\right] \\
&= \arg\max_{w_l} P(w_l|X) \\
&= \arg\max_{w_l} P(w_l|\hat{W}_{\neg l})p(X|w_l)
\end{aligned}$$

where $p(X|w_l)$ is called the "likelihood" that word w_l produces audio file X, and $P(w_l|\hat{W}_{\neg l})$ is the prior probability of word w_l given all of the other words in the transcription, $\hat{W}_{\neg l} = [\hat{w}_1, \ldots, \hat{w}_{l-1}, \hat{w}_{l+1}, \ldots, \hat{w}_L]$.

In automatic speech recognition, the likelihood is approximated by a parameterized function called the "acoustic model," $\hat{p}(X|w_l)$, and the prior probability is approximated by a lookup table called the "language model," $\hat{P}(w_l|\hat{W}_{\neg l})$. The goal of speech recognizer training is to estimate these two functions with the smallest possible *bias* and the smallest possible *variance*:

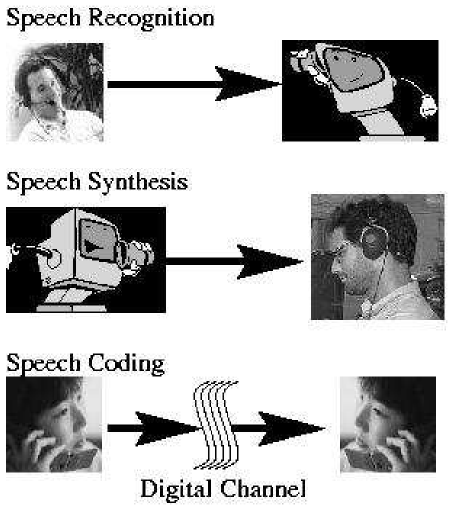

Figure 1: Speech recognition converts an audio file into text. Speech synthesis converts text into audio. Speech coding converts audio into an efficient digital code, suitable for efficient storage or transmission.

1. *Bias:* The true posterior probability of the correct word, given the audio file, should be 1.0. Here is a recognizer that can transcribe the correct word with nearly zero probability of error: (1) play the audio file to a human being, (2) ask him or her to write down what she hears.[1] Therefore we wish to design our recognizer so that, in the limit, as the number of training tokens goes to infinity, it can learn to transcribe with zero error. This can be done by guaranteeing that, in the limit as the training size grows to infinity, the functions $\hat{p}(X|w)$ and $\hat{P}(w_l|W_{\neg l})$ are able to learn the real values of the true underlying distributions $p(X|w)$ and $P(w_l|W_{\neg l})$ with no errors anywhere, regardless of how messy those distributions might be. Parameterized functions that have this property—the property of being able to approximate any arbitrary messy distribution—are called "universal approximators."

2. *Variance:* we want to make sure that the error rate estimated on training data is, as nearly as possible, the same as the error rate estimated on test data. There are basically two ways to address this problem.

   (a) First, make sure that the training database is as large as possible. In practice, this means that you should choose the number of Gaussian kernels per state (K in Eq. 5) to match the number of available training tokens. There are two useful methods:

      i. Before you have the system designed, you might want to estimate how much training data you need. In this case, a good rule of thumb is the "rule of 5:" the number of measurements in the training data should be at least five to ten times the number of trainable parameters. Consider a relatively typical large-vocabulary speech recognizer in which each phone model (each monophone, or triphone, or prosody-dependent allophone model) is represented by an HMM containing N states, each state containing K Gaussians. Each Gaussian has one trainable mean per feature dimension, and one trainable variance per feature dimension, so in order to train this system, we need $(5-10) \times N \times K \times 2 \approx (10-20)NK$ frames of data for each phone model.[2] A typical phone is about 50ms, or five frames (some are longer, some shorter), so we need about $2NK - 4NK$ examples of each phone in order to ensure reliable training. So, for example, to train a 3-state HMM with one Gaussian per state, you need at least 6-12 examples of each phone; for a 128-mixture Gaussian, you need at least 374-648 examples of every phone.

[1] Actual error rates of human transcribers are around 0.05% for phone numbers, and around 4% for conversational telephone speech; error rates of the best automatic systems are now around 4× higher for both tasks.

[2] Duda, Hart, and Stork would say $10NKD$ where D is the feature dimension [7], but cross-validation experiments suggest that $10NK$ seems to be enough in ASR systems.

ii. Once you have started training, you should verify the "rule of thumb" by testing the system on validation data. The best way to train a 128-Gaussian system is by first training systems with 1, 2, 4, 8,..., 64 Gaussians. So, test each of those systems on held-out "validation data;" your final system will be the one that performs best on validation data.

(b) Second, use constrained or regularized learning. Regularized learning compensates for small training datasets by penalizing parameter vectors θ that seem intuitively implausible or undesirable; constrained learning compensates for small training datasets by simply refusing to consider implausible parameter vectors. Typical examples of constrained learning include variance floor[21] and MLLR adaptive learning [14]; typical examples of regularized learning include MAP adaptation [12] and large-margin HMMs [10].

The rest of this section will demonstrate that the N-gram language model and the mixture Gaussian HMM are universal approximators (in other words, that, as the amount of training data becomes infinite, it becomes possible to train an HMM with zero *bias*). The next section will talk a little bit about the *variance* of the HMM estimate—in particular, about the great pains that we take, during training of the acoustic model, to make sure that the acoustic model converges to a reasonable set of parameters.

An N-gram approximates the probability of a word sequence $W = [w_1, \ldots, w_L]$ as

$$P(W) = P(w_1)P(w2|w_1)\ldots P(w_L|w_1,\ldots,w_{L-1}) \quad (2)$$

$$\approx \prod_{l=1}^{L} \hat{P}(w_l|w_{l-N+1},\ldots,w_{l-2},w_l) \quad (3)$$

Maximum likelihood estimates of the N-gram probability are given by

$$\hat{P}^*(w_4|w_1,w_2,w_3) = \frac{N(w_1,w_2,w_3,w_4)}{N(w_1,w_2,w_3)}$$

where $N(\cdot)$ is the frequency count of event $\cdot$ in a training database. As $N(\cdot)$ approaches infinity, the *variance* of the estimate approaches zero; if $N > L$ (if the N-gram knows all available context), then Eq. 2 equals Eq. 3 exactly, and therefore the *bias* of the estimate is also zero. Unfortunately, for a fixed-size training dataset, there is a tradeoff between large N (the size of the history) and large $N(\cdot)$ (the frequency of the history). With ten hours of speech, you can reliably train a bigram ($N = 2$); with a billion words of data, you can reliably train a 4-gram ($N = 4$); using the entire internet as their database, google has trained and released (via LDC) a set of 5-gram language models ($N = 5$). Even $N = 5$ is not long enough to include all of the context information available to human listeners. Longer context can be incorporated in a few different ways, e.g., using word histories tagged with phrase structure [4] or argument structure [20], or using variable-N-grams [19].

A hidden Markov model approximates the probability of the waveform X as follows. First, you should know that an audio file on your computer is actually a series of samples, $x[0], x[1], \ldots$, representing the values of the microphone voltage once every $1/F_s$ seconds. For the sake of example, let's suppose that $F_s = 16000$ samples/second (F_s is typically between 8000 and 44100 samples/second). Suppose that we re-arrange the waveform samples as follows:

$$X = [\vec{x}_0, \vec{x}_1, \ldots, \vec{x}_{T-1}]$$

where each of the vectors is a "frame" containing W consecutive speech samples, with an overlap between neighboring frames of $W - S$ samples:

$$\vec{x}_t^T = \begin{bmatrix} x[tS] \\ x[tS+1] \\ \vdots \\ x[tS+W-1] \end{bmatrix}$$

The hidden Markov model represents $p(X|W)$ by dividing each word w_l into a series of sub-word states, so that

$$\hat{p}(X|W) = \sum_Q \prod_{t=0}^{T-1} \hat{p}(\vec{x}_t|q_t,\theta)\hat{P}(q_t|q_{t-1},W,\theta) \quad (4)$$

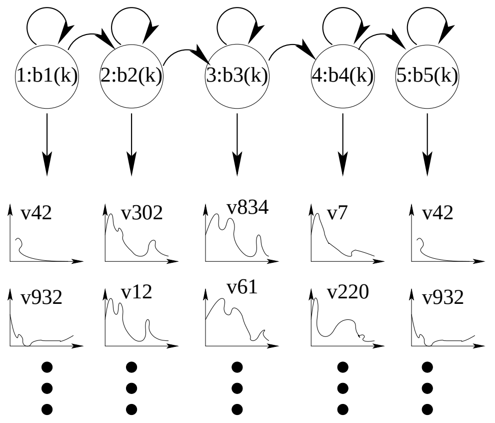

Figure 2: Ideally, the states of the hidden Markov model should capture all relevant information that might influence the acoustic spectrum.

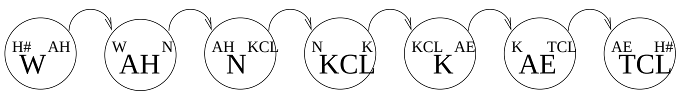

Figure 3: In practice, it is common, at least, for an HMM to explicitly represent the phone currently being spoken, as well as its triphone context: the immediately preceding phone, and the immediately following phone [13]. In this example, the HMM represents the word sequence "one cat;" the six phones composing this word sequence are represented here in ARPABET notation.

where q_t is the sub-word state at time t, and the sum is over all possible state sequences $Q = [q_0, \ldots, q_{T-1}]$ that might be reasonable representations of sentence W. Eq. 4 can be an exact equation, with no approximations, if the state q_{t-1} summarizes all information about the context that is necessary to determine the state q_t, and if the state q_t summarizes all information about the context and the sentence that is necessary to determine $\vec{x}_t$ (Fig. 2). In order for these things to happen, we need to allow an infinite number of states. In particular, we need to allow:

1. The state variable q_t needs to summarize all relevant context information. Some types of context information that are commonly used in ASR include: the identity of the phoneme currently being produced (monophone-based recognizer), and the identities of the immediately preceding and immediately following phones (triphone-based recognizer: Fig. 3). Less commonly used types of context, types of context that tend to reduce the word error rate by small but statistically significant amounts, include the identity of the word being spoken (word-dependent phone models), the prosodic context (prosody-dependent speech recognizer), and the speaker ID (speaker-dependent pronunciation models.) In general, the larger the number of context-dependent phone models you can train, the lower the word error rate of the recognizer, subject to the rule-of-five described earlier.

2. It needs to be possible for q_t to be different from q_{t-1}, so that we don't have to pretend that dynamic spectra are static. This means that the number of states per phone model should also be allowed to increase toward infinity, as one acquires more and more training data. Few systems use more than three states per phone, though several systems have attempted some type of continuous interpolation between neighboring states.

Finally, in order for the state q_t to summarize all relevant information about the acoustic frame $\vec{x}_t$, it is necessary that the probability density $\hat{p}(\vec{x}_t|q_t)$ should itself be a universal approximator. In practice, we use the mixture Gaussian universal approximator. The mixture Gaussian is able to represent any smooth function as a series of localized Gaussian "bumps" (Fig. 4). The mathematical form of the mixture Gaussian

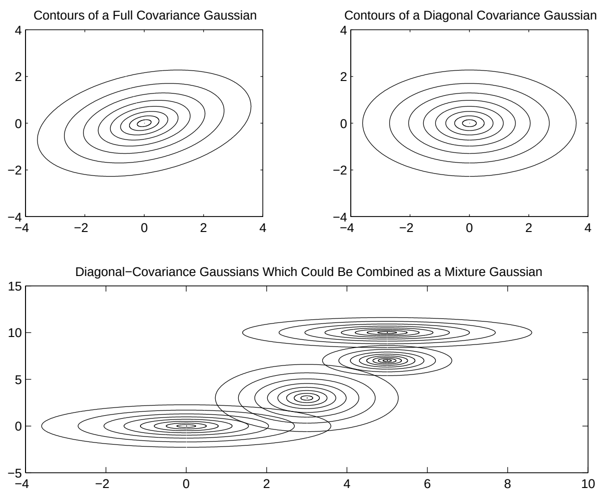

Figure 4: Contour plots of the Gaussian, diagonal-covariance Gaussian, and mixture Gaussian probability density functions. The mixture Gaussian can approximate any smooth function as a series of localized bumps, hence, as $K \rightarrow \infty$, it is a universal approximator.

is:

$$\hat{p}(\vec{x}_t|q) = \sum_{k=1}^{K} c_{qk} \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{qkd}^2}} e^{-\frac{(o_{td}-\mu_{qkd})^2}{2\sigma_{qkd}^2}} \tag{5}$$

where o_{td} is the dth component of the vector $\vec{o}_t$, and the parameters c_{qk}, μ_{qkd}, and σ_{qkd}^2 have been learned from training data. The vector $\vec{o}_t$ is a nonlinear transformation of the vector of speech samples, $\vec{x}_t$, designed to keep all of the relevant information. The MFCC (mel-frequency cepstral coefficients [6]), for example, are computed as

$$\vec{o}_t = LC \log\left(H \left|F\vec{x}_t\right|.^2\right) \tag{6}$$

where F is a $W \times W$ Fourier transform matrix, H is an $N_C \times W$ mel-frequency filterbank matrix, C is an $N_L \times N_C$ discrete cosine transform matrix, and L is a $D \times N_L$ liftering matrix. Each of these transforms represents a different aspect of human audio perception.

In summary, the goal of a "universal approximator" is to quantify our lack of understanding. Without universal approximators, we don't know how bad our models are. With a universal approximator, the *bias* and *variance* of the approximator are explicitly traded off, one against the other, by the parameters N, N_H, N_Q, K, and D, and N_F:

1. N_F is the size of the training dataset.

2. N, the history length of the N-gram. Bias of the N-gram goes to zero when $N \geq L$, but variance is proportional to N/N_F.

3. N_H, the number of different HMMs used to represent speech (e.g., each HMM might be a context-dependent triphone model). As $N_H \rightarrow \infty$, bias goes to zero, but variance is proportional to N_H/N_F.

4. N_Q, the number of states in each phone model HMM. Bias goes to zero when N_Q is greater than the number of frames ($N_Q \geq T$), but variance is proportional to N_Q/N_F.

5. K, the number of Gaussians in each state. Bias goes to zero as $K \rightarrow \infty$, but variance is proportional to K/N_F.

6. D, the dimension of the feature vector. A feature encoding such as MFCC (Eq. 6) can be designed so that bias is zero when the feature dimension is equal to the frame skip ($D \geq S$), but variance is a monotonically increasing function of D/N_F.

Notice that the only parameters that need to go to infinity are N_H and K. The other three requirements ($N \geq L$, $N_Q \geq T$, $D \geq S$) are nearly satisfied by large-vocabulary speech recognizers with variable-N-gram language models.

2 Acoustic Model Training: Overview

The first time that you train a speech recognizer, you're likely to spend most of your time training the acoustic models. Future lectures will come back and work through these steps in more detail, using an example script called train.pl; in this lecture, I'd like to just introduce the steps, and then talk about the different types of information that you need, from your database, in order to get started.

Remember that an acoustic model is just a parameterized function, $p(x_t|q)$, where q is the phone state, and x_t is the speech feature vector (PLP or MFCC or Tandem or whatever). In a complete system, $p(x_t|q)$ is a mixture of Gaussians, but it starts out simple:

1. First, we create a universal background unimodal Gaussian (UBUG), $p(x_t)$. That previous sentence is a high-falutin' way of saying that we should compute the mean and variance of each feature. The HTK command line tool `HCompV` does this.

2. Second, we train monophone models (one HMM for each phoneme of the language, typically about 40 phones for a single-language recognizer) with Gaussian PDFs.

   (a) Use your dictionary, together with your non-time-aligned orthographic transcriptions, in order to produce a non-time-aligned phone transcription. `HLEd` does this.

   (b) Create prototype HMMs for each phone. In order to make sure that the initial settings of the likelihood functions are not completely nonsense, we initially just copy the UBUG into the definition for each phone. `HHEd` does this.

   (c) Use the EM algorithm (expectation maximization) to re-estimate the parameters of each phone model. Much has been written about the EM algorithm, but here are the references I like best: Baum proved it [1, 2], Liporace applied it to Gaussians [16], Juang et al. applied it to mixture Gaussians [11], Levinson and Rabiner wrote the best tutorial for those who want to know how it works but not why ([15], republished as [18]), and Bilmes wrote the best tutorial for those who want to know why it works [3]. `HERest` does this.

   (d) Fifth, it is often useful to time-align the phone transcriptions, and in particular, to let the recognizer tell you if some of your talkers are using non-standard pronunciations. This can be done using `HVite`.

   (e) Sixth, you should re-re-estimate your acoustic models using the time-aligned and corrected transcriptions. `HRest` or `HERest` can be used for this purpose.

   (f) Speaker-Adaptive training would go about here, but we have not yet used it at Illinois. Speaker-Adaptive training [8, 17] compensates for speaker differences during acoustic model training: each speaker's training data is linearly transformed so that it more closely resembles the training data for a "prototype" speaker. In this way, acoustic models are made more precise, because the Gaussian doesn't have to model inter-speaker variability—instead, inter-speaker variability is handled by a separate speaker normalization step.

3. Triphones! Triphones are context-dependent acoustic models, e.g., the /t/ in "this tree" can now be different from the /t/ in "that tip." We use the following notation: a /t/, preceded by an /s/, and followed by an /r/, is written /s-t+r/. Notice that this is a type of /t/; the /s-/ and /+r/ bits specify only the phonetic context. At Illinois we also usually specify prosodic context (lexical stress, phrase position), but this is hard to do in a corpus-independent way.

   (a) First, the transcription is converted from a time-aligned monophone to a time-aligned triphone transcription, using `HLEd`.

   (b) Second, triphone models are created for every distinct triphone. To start with, each triphone model duplicates the PDF of its corresponding monophone; this cloning is done using `HHEd`.

   (c) Use `HRest` and/or `HERest` to re-train.

   (d) You probably don't have enough data to train distinct acoustic models for each and every three-phone sequence in the English language. `HHEd` can be used to cluster the triphones, and `HERest` or `HRest` can re-estimate the clustered triphones.

4. Now you are finally ready to convert the Gaussian PDFs into Mixture Gaussian PDFs! This is done using several iterations of `HHEd` and `HERest`, one after the other. This process must be done slowly, because any mixture Gaussian with $K > 1$ suffers from spurious and undesirable global optimum parameter settings (that phrase, translated into plain English: if you try to learn a 128-component mixture Gaussian all at once without proper initialization, the training algorithm will learn a set of parameters that work really well for the training data and really badly for anything else, usually including at least one nearly-zero variance parameter). In order to avoid these spurious global optima, we usually upmix slowly, e.g., going from one Gaussian to two, then to four, and so on, checking the variances at each step to make sure no variance parameter is getting too small.

5. Finally, HMMs should be discriminatively trained, using `HMMIRest`. Discriminative training is just like EM training, but with a different optimality criterion—one that better predicts the final error rate of the recognizer. In order to avoid over-fitting the recognizer to the training database, the strict error-rate criterion is regularized, with regularization constant adjusted to give you the degree of generality you require.

3 Acoustic Model Training Using HTK

At this point in lecture, we will look at a sample shell script or perl script that goes through all of the steps above in order to train an HMM (train_spid.sh trains a speaker-ID system, train.pl trains speech recognition systems). The rest of this section discusses some of the text files required to do so.

3.1 Transcriptions: Master Label File

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.

3.2 Database File Listing: 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.

3.3 HMM Listing: PHF file

Almost all HTK tools require you to list the names of the acoustic models you wish to train. I have invented the extension "PHF" (phone file) for this type of file, just so that I can read my bash scripts a little more easily.

Usually, you have one acoustic model per monophone, or one acoustic model per triphone, therefore a PHF file for the automatic recognition of re-iterant speech (speech where your subject says "ma ma ma ma ma ma") might look like:

```
a
m
sil
```

3.4 Dictionaries

A dictionary is a mapping from words to phones. There are three important differences between dictionaries:

1. What phone code is used? That is, what symbols are used to represent the phonemes?
2. Do words have only one pronunciation each, or do they have alternate pronunciations? If alternate pronunciations are available, is there any way to measure the probability of each alternate pronunciation?
3. How many words are there? In particular, does the dictionary include each of the following types of pseudo-words (and do you want it to):
   - Word fragments. Most training databases include utterances of word fragments, such as "do-" for "don't." It is possible for a recognizer to recognize word fragments during the test phase, but it's actually really difficult, because allowing a word to break off in the middle opens up lots of possibilities for errors. Therefore, in practice, most speech engineers start with a dictionary that includes no word fragments (though the ISLEX dictionary includes word fragments, http://www.isle.uiuc.edu/dict), then find out which word fragments exist in the training database, and estimate the pronunciations of those word fragments using a dictionary-based text-to-phone mapping such as http://mickey.ifp.uiuc.edu/speechWiki/index.php/Phonetic_Transcription_Tool.
   - Filled pauses. "UH" and "UM" should certainly be distinguished [5]. Some dictionaries also distinguish "EH," and other types of non-words.
   - Non-word sounds. For example, I usually include entries of the following type, where the first column are the "word" labels, and the second column are the "phone" labels. Each such "phone" gets its own hidden Markov model. Lines of this type should be included in the dictionary for each type of non-word event that is transcribed in your training data, though it is often necessary to cluster the non-speech events into acoustically-similar categories.

     ```
     LAUGH laugh
     COUGH cough
     MOUTHNOISE mouthnoise
     NOISE noise
     SILENCE sil
     UNKOWN\_WORD unk
     UH pv
     UM pn
     ```

There are at least three phone codes worth mentioning:

1. The international phonetic alphabet (IPA, http://www.arts.gla.ac.uk/IPA/ipachart.html) should be considered authoritative. Unfortunately, you can only use IPA symbols to name your HMMs if all of your software understands Unicode (http://unicode.org); most modern tools do (perl, python, ruby, xterm), but HTK does not.

2. Worldbet (http://cslu.cse.ogi.edu/publications/ps/hieronymus_att_worldbet.ps.gz) is an ASCII encoding of every symbol in the IPA, hence it is useful for software (like HTK) that only understands ASCII. Unfortunately, some of the symbols in worldbet cause HTK to choke, e.g., any phone symbol starting with a digit should be prepended with a letter (say, "n") before being used as the name of an HMM in HTK. Example (# is a word boundary, . is a syllable boundary)

   ```
   administration #@d.mI.nI.stra.S&n#
   ```

3. ARPABET (http://en.wikipedia.org/wiki/Arpabet) is a case-independent, alphabetic encoding of the phones of American English. HTK works really well with ARPABET, but of course, ARPABET is only useful if you're designing a speech recognizer for American English. Example:

   ```
   administration ae d m ih n ih s t r ey sh ih n
   ```

3.5 Master Model Files

The hidden Markov models are stored in master model files. See HTK Book for many good examples, or makemmf.rb for ruby code that generates some. Here is a pretty standard example, generated by makemmf.rb, with two phones called /m/ and /aa/, each with $N_Q = 3$ states, $K = 1$ Gaussian per state, and (for example purposes) $D = 8$ dimensions per Gaussian:

```
~o <VecSize> 8 <PLP_E_D_A>
~h "aa"
<BeginHMM>
 <NumStates> 5
 <State> 2
   <Mean> 8
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 8
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 3
   <Mean> 8
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 8
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 4
   <Mean> 8
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 8
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <TransP> 5
 0.0 1.0 0.0 0.0 0.0
 0.0 0.7 0.3 0.0 0.0
 0.0 0.0 0.7 0.3 0.0
 0.0 0.0 0.0 0.7 0.3
 0.0 0.0 0.0 0.0 0.0
<EndHMM>
~h "m"
<BeginHMM>
 <NumStates> 5
 <State> 2
   <Mean> 8
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 8
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

```
 <State> 3
   <Mean> 8
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 8
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <State> 4
   <Mean> 8
     0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
   <Variance> 8
     1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
 <TransP> 5
 0.0 1.0 0.0 0.0 0.0
 0.0 0.7 0.3 0.0 0.0
 0.0 0.0 0.7 0.3 0.0
 0.0 0.0 0.0 0.7 0.3
 0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

References

[1] Leonard E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. Am. Math. Soc.*, 73:360–363, 1967.

[2] Leonard E. Baum and George R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27(2):211–227, 1968.

[3] Jeff A. Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report TR-97-021, International Computer Sciences Institute, Berkeley, CA, 1997.

[4] Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech and Language*, 14(4):283–332, 2000.

[5] Herbert H. Clark and Jean E. Fox Tree. Using *uh* and *um* in spontaneous speaking. *Cognition*, 84:73–111, 2002.

[6] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-28(4):357–366, August 1980.

[7] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* John Wiley & Sons, New York, 2001.

[8] Mark J. F. Gales and Philip Woodland. Speech recognition the HTK way: Tutorial session. In *Proc. ICASSP*, 2006.

[9] Frederick Jelinek. Continuous speech recognition by statistical methods. *Proc. IEEE*, 64(4):532–556, 1976.

[10] Hui Jiang, Xinwei Li, and Chaojun Liu. Large margin hidden markov models for speech recognition. *IEEE Trans. Audio, Speech and Language*, 14(5):1584–1595, 2006.

[11] 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.

[12] Chin-Hui Lee, Chih-Heng Lin, and Bing-Hwang Juang. A study on speaker adaptation of the parameters of continuous density hidden markov models. *IEEE Trans. Speech and Audio Processing*, 39(4):806–814, 1991.

[13] Kai-Fu Lee. Context-dependent phonetic hidden markov models for speaker-independent continuous speech recognition. *IEEE Trans. on Acoustics, Speech, and Sig. Proc.*, 38, 1990.

[14] C. J. Leggetter and P. C. Woodland. Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Comput. Speech Lang*, 9:171–185, 1995.

[15] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell System Technical Journal*, 62(4):1035–1074, Apr. 1983.

[16] Louis A. Liporace. Maximum likelihood estimation of multivariate observations of Markov sources. *IEEE Trans. on Information Theory*, 28(5):729–734, 1982.

[17] Spyros Matsoukas, Jean-Luc Gauvain, Gilles Adda, Thomas Colthurst, Cia-Lin Kao, Owen Kimball, Lori Lamel, Fabrice Lefevre, Jeff Z. Ma, John Makhoul, Long Nguyen, Rohit Prasad, Richard Schwartz, Holger Schwenk, and Bing Xiang. Advances in transcription of broadcast news and conversational telephone speech within the combined EARS BBN/LIMSI system. *IEEE Trans. Audio, Speech and Language*, 14(5):1541–1556, 2006.

[18] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 1989.

[19] Manhung Siu and Mari Ostendorf. Variable n-grams and extensions for conversational speech language modeling. *IEEE Trans. Speech and Audio Processing*, 8(1):63–75, 2000.

[20] W. Wang, A. Stolcke, and M.P. Harper. The use of a linguistically motivated language model inconversational speech recognition. In *Proceedings of ICASSP*, 2004.

[21] 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.