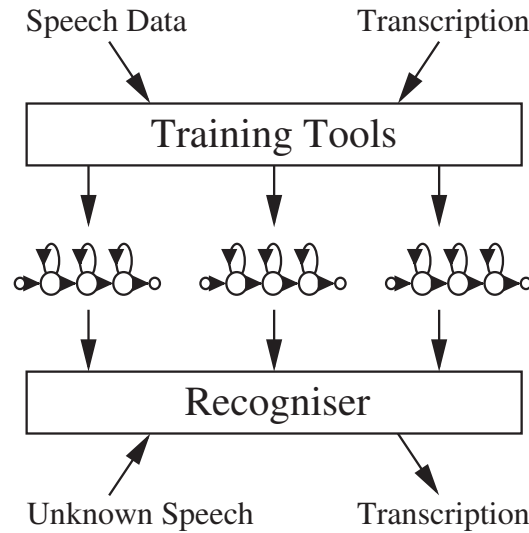


Speech Processing (Module CS5241)

Lecture 8 – Introduction to HTK



SIM Khe Chai

Introduction to HTK

HTK – Hidden Markov Model Toolkit – is a toolkit for building HMM-based acoustic models as well as statistical back-off n -gram language models for automatic speech recognition. It consists of a set of commandline tools written in C language to construct various components of a speech recognition system.

HTK is an open-source research toolkit available for free download from:

<http://htk.eng.cam.ac.uk/>

HTK consists of the following:

- **HTKLib**: a set of library modules for building the HMM tools
- **HTKTools**: a set of commandline tools for building HMM models
- **HLMLib**: a set of library modules for building the HLM tools
- **HLMTools**: a set of commandline tools for building n -gram language models
- **HTKBook**: a reference document comprise basic theory of HMM and n -gram LM, usage manual for various tools and tutorial examples

Compiling HTK (linux)

- Download latest version (3.4): `HTK-3.4.tar.gz`
- Uncompress the tar ball: `tar zxvf HTK-3.4.tar.gz`
- Move to the htk directory: `cd htk`
- Configure the make file: `./configure --enable-trad-htk --prefix=$PWD`
- Compile HTK: `make all`
- The binaries for various tools are found in: `bin.linux/`

The 34 command line tools are:

HTKTools	HCompV HDMan HHed HLEd HLStats HParse HRest HSGen HSmooth HBuild HCopy HERest HInit HList HLRescore HMMIRest HQuant HResults HSLab HVite
HLMTools	Cluster HLMCopy LAdapt LFoF LGList LLink LNewMap LPlex LBuild LGCopy LGPrep LMerge LNorm LSubset

Getting Usage Information

To obtain the usage information for each HTK tool, just execute the tool without command line arguments. Example: HSGen

```
USAGE: HSGen [options] latticeFile dictFile

Option                                Default

-l      Include line numbers          off
-n N    Number of sentences to generate 100
-q      Suppress sentence output       off
-s      Compute grammar stats          off
-A      Print command line arguments   off
-C cf   Set config file to cf          default
-D      Display configuration variables off
-S f    Set script file to f           none
-T N    Set trace flags to N           0
-V      Print version information       off
```

Standard HTK Options

There are several standard HTK options that are applicable to most HTK tools:

- A: Print command line arguments
 - Useful when execution output is piped to a log file
- V: Print version information
 - Prints out the version of each HTK module used to compile the tool
- D: Display configuration variables
 - Prints out the configuration settings as supplied by the `-C` option
- T: Set trace flags
 - Set different tracing levels (for debugging & info)
- S: Set script file
 - Set script files for batch processing (more info later)
- C: Set config file
 - Set config files to customise settings (more info later)

HTK Configuration Files

The `-C` standard option allows configuration files to be specified. Each line in the config file specifies the value for a config variable. Each line can be one of the following forms:

- `<CONFIG_NAME> = <CONFIG_VALUE>`
- `<CONFIG_MODULE>:<CONFIG_NAME> = <CONFIG_VALUE>`

`#` marks the beginning of comments (per line). Empty lines are ignored.

Example:

```
# A config file for HMM training

TARGETKIND      = MFCC_0_D_A_T_Z      #   Basic parameterisation

HFB:TRACE       = 1

# HMODEL:SAVEBINARY = T
```

Batch Processing using Script Files

The `-S` standard option allows script files to be used. HTK tools simply appends the supplied command line arguments with all the lines in the script file.

Example:

```
HCopy -C config 1.wav 1.mfcc 2.wav 2.mfcc ...
```

The above example uses HCopy to convert wave files to MFCC parameter files. Instead of listing all the source and target files in the command line explicitly, it is possible to supply a script file that stores the filenames. Example the files in `files.scp`:

```
1.wav 1.mfcc  
2.wav 2.mfcc  
...
```

or

```
1.wav  
1.mfcc  
2.wav  
2.mfcc  
...
```

Command using script file: `HCopy -C config -S files.scp`

Building A Continuous Digit Recognition System using the HTK

- Preparation
 - Feature Extraction (HCopy)
 - Pronunciation Dictionary (HDMAN)
 - Label Transcriptions (HLEd)
- HMM Training
 - HMM Prototype Definition
 - Flat start (HCompV)
 - Baum-Welch Re-estimation (HERest)
 - GMM building using iterative mixture splitting (HHEd)
- HMM Evaluation
 - Decoding network construction (HBuild)
 - Viterbi decoding (HVite)
 - Performance evaluation (HResults)

Preparing the Vocabulary & Lexicon

The word list:

digit.wordlist
<s>
</s>
ZERO
ONE
TWO
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE

The pronunciation dictionary:

digit.dictionary	digit.dictionary (cont'd)
<s> sil	FOUR f ao r sp
</s> sil	FIVE f ay v sil
ZERO z iy r ow sil	FIVE f ay v sp
ZERO z iy r ow sp	SIX s ih k s sil
ONE w ah n sil	SIX s ih k s sp
ONE w ah n sp	SEVEN s eh v ah n sil
TWO t uw sil	SEVEN s eh v ah n sp
TWO t uw sp	EIGHT ey t sil
THREE th r iy sil	EIGHT ey t sp
THREE th r iy sp	NINE n ay n sil
FOUR f ao r sil	NINE n ay n sp

Building A Digit Loop (HBuild)

```
Command: HBuild -t '!SENT_START' '!SENT_END' digit.wordlist digit.loop
```

```
N=14    L=23
I=0     W=!NULL
I=1     W=<s>
I=2     W=ZERO
I=3     W=ONE
...
J=0     S=0    E=1    l=0.00
J=1     S=1    E=2    l=-2.30
J=2     S=1    E=3    l=-2.30
...
```

- There network file may include some optional header information:
 - version
 - vocabulary filename
 - user comments
- All the words in the network generated by HBuild have the same language model scores
- Network can also be generated using grammar rules (HParse) – see previous lecture

Generate Digit Sequences (HSGen)

Command: HSGen -l -n 500 digit.loop digit.dictionary > digit.sentences

```
1. ONE ONE
2. FOUR
3. SIX FOUR
4. NINE
5. ZERO
6. THREE TWO
7. FIVE
8. SEVEN SIX FOUR ZERO FOUR SEVEN TWO
9. NINE
10. TWO
...
```

Collecting Voice Data

Given the generated sentences, collect speech data for these sentences. HTK comes with a simple speech recording tool with Graphical User Interface (GUI) for data collection – HSLab. It is not the best speech recording software. Since HTK support various audio formats including the MS Windows WAVE format, any general purpose audio recording software can be used for voice data collection.

Audio file format supported by HTK are:

NOHEAD	no head at all	SDES1	Sound Designer I format
ALIEN	alien ie unknown	SUNAU8	Sun 8 bit MU law .au format
HTK	the preferred HMM Toolkit format	OGI	Oregon Institute format
TIMIT	the TIMIT database	WAV	Microsoft WAV format
NIST	the NIST SPHERE format	ESPS	Entropic format
SCRIBE	the UK SCRIBE database	ESIG	Entropic format
AIFF	Apple audio interchange format		

Feature Extraction (HCopy)

Command: HCopy -C mfcc.cfg -S digit.wav2mfcc.scp

mfcc.cfg:

digit.wav2mfcc.scp:

```
SOURCEFORMAT=HTK
TARGETKIND=MFCC_E_D_A_T
TARGETRATE=100000.0
WINDOWSIZE=256000
USEHAMMING=T
PREEMCOEF=0.97
NUMCHANS=26
CEPLIFTER=22
NUMCEPS=12
ENORMALISE=T
```

```
data/sent001.wav mfcc/sent001.mfcc
data/sent002.wav mfcc/sent002.mfcc
data/sent003.wav mfcc/sent003.mfcc
data/sent004.wav mfcc/sent004.mfcc
data/sent005.wav mfcc/sent005.mfcc
data/sent006.wav mfcc/sent006.mfcc
data/sent007.wav mfcc/sent007.mfcc
data/sent008.wav mfcc/sent008.mfcc
data/sent009.wav mfcc/sent009.mfcc
...
```

Feature Types Supported by HTK

HTK supports:

- LPC – Linear Prediction Coefficients
- MELSPEC – Mel-frequency spectral magnitude
- FBANK – Log filter bank coefficients
- MFCC – Mel Frequency Cepstral Coefficients
- PLP – Perceptual Linear Prediction coefficients

Additional qualifiers:

Qualifier	Meaning
_E	energy term
_0	zeroth coefficients
_D	first differential
_A	second differential
_T	third differential
_Z	utterance-based mean normalisation

Viewing Parameter Files (HList)

For efficiency, parameter files are stored in binary format. To view its content, use HList

Example: HList -h data.mfcc

```
----- Source: data.mfcc -----
Sample Bytes:  18      Sample Kind:  MFCC_E_0
Num Comps:    9       Sample Period: 10000.0 us
Num Samples:  3       File Format:   HTK
----- Samples: 0->2 -----
0:   -18.548  -5.480   7.343   3.620  -5.057  -4.688  -3.676  45.615  -0.151
1:   -17.970  -4.205  -0.815  -3.309   4.400   2.110  -2.763  44.061  -0.151
2:   -17.355  -3.660   0.901   1.393   0.708   5.566  -4.706  43.115  -0.151
----- END -----
```

Actual file format:

Byte	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Format	integer				integer				short		short		float				...
Data	No. samples				Samp. Period				Samp. Size		Samp. Kind		Data point				...

Note, HTK stored data in 'big endian' format. Can specify NATURALREAD and NATURALWRITE to override defaults.

Physical vs. Logical Files

HTK supports **physical** and **logical** files for several file types: audio files, feature parameter files, label (transcription) files, HMM definition files, etc.

- **Physical files:** actual files stored on disks
- **Logical files:** pointer (link) to physical files or exists as part of a larger file

Specifying logical input parameter files:

- <LOGICAL_FILENAME>=<PHYSICAL_FILENAME>

Examples:

```
user1_data1.mfcc=/data/user1/data1.mfcc
user1_data2.mfcc=/data/user1/data2.mfcc
user1_data3.mfcc=/data/user1/data3.mfcc
user2_data1.mfcc=/data/user2/data1.mfcc
user2_data2.mfcc=/data/user2/data2.mfcc
```

Logical File Segmentation

Voice Activity Detection (VAD) is an important stage for speech processing. This process divides audio data into speech and non-speech segments. Each segments can be stored as individual physical files. However, this may result in many small physical files which can degrade I/O performance and storage efficiency.

Alternatively, keep the original unsegmented file and specify logical files which point to specific regions within the original file:

Examples:

```
data1_segment1.mfcc=/data/data1.mfcc[1,300]
data1_segment2.mfcc=/data/data1.mfcc[1000,1500]
data1_segment3.mfcc=/data/data1.mfcc[1800,2000]
data1_segment4.mfcc=/data/data1.mfcc[2100,3000]
data2_segment1.mfcc=/data/data2.mfcc[1,600]
data2_segment2.mfcc=/data/data2.mfcc[2000,2300]
data2_segment3.mfcc=/data/data2.mfcc[4800,5500]
```

Preparing Transcriptions

In order to train the acoustic models, the word-level transcriptions need to be supplied for each audio file. HTK uses the **base name** of the supplied audio/data file to search for the transcription file. If logical files are used, base names are derived from the logical file names. Then, HTK searches for label files in the specified **label directory** using the specified **label extension**. These are specified using the options:

- 'L' specifies the label directory (default is current directory, '.')
- 'X' specifies the label extension (default is 'lab')

Example:

Data file	Directory	Extension	Label file
/user1/data1.mfcc	labdir	lab	labdir/data.lab
user1_data1.mfcc=/user1/data1.mfcc	.	txt	./user1_data.txt

- The label file is a plain text with one word per line.

Master Label File (MLF)

HTK supports logical label files using the Master Label File (MLF).

<pre>#!MLF!# "logical1.lab" WORD1 WORD2 . "*/logical2.lab" WORD1 WORD2 . "*logical3.lab" WORD1 WORD2 .</pre>	<pre>→ logical1.mfcc → data/user1/logical2.mfcc → user1_logical2.mfcc</pre>
--	---

MLF combines multiple transcription files into a single file. MLF files are loaded using the '-l' option.

Creating Transcriptions in MLF format

```
sent001 ONE TWO THREE
sent002 FOUR FIVE
sent003 SIX SEVEN
```



```
BEGIN{ print "#!MLF!#"; }
{
    printf("\n*/%s.lab\n\n", $1);
    for (i=2; i<=NF; i++)
        print $i;
    print ".";
}
```



```
#!MLF!#
"/sent001.lab"
ONE
TWO
THREE
.
"/sent002.lab"
FOUR
FIVE
.
"/sent003.lab"
SIX
SEVEN
.
```

Manipulating label files using (HLEd)

HLEd is a label editor. It:

- takes in label files or MLF file
- manipulate/edit the label files
- store output in MLF format

Typical commands:

- `HLEd -l <DIR> -X <EXT> -i <OUTPUT_MLF> <EDIT_FILE> <INPUT_LABELS> ...`
- `HLEd -l <DIR> -X <EXT> -i <OUTPUT_MLF> <EDIT_FILE> <INPUT_MLF> ...`
- `HLEd -l <DIR> -X <EXT> -i <OUTPUT_MLF> -I <INPUT_MLF> <EDIT_FILE>`

HTK options:

- 'l' option specifies the label directory
- 'X' option specifies the label extension
- 'I' option specifies the input MLF file

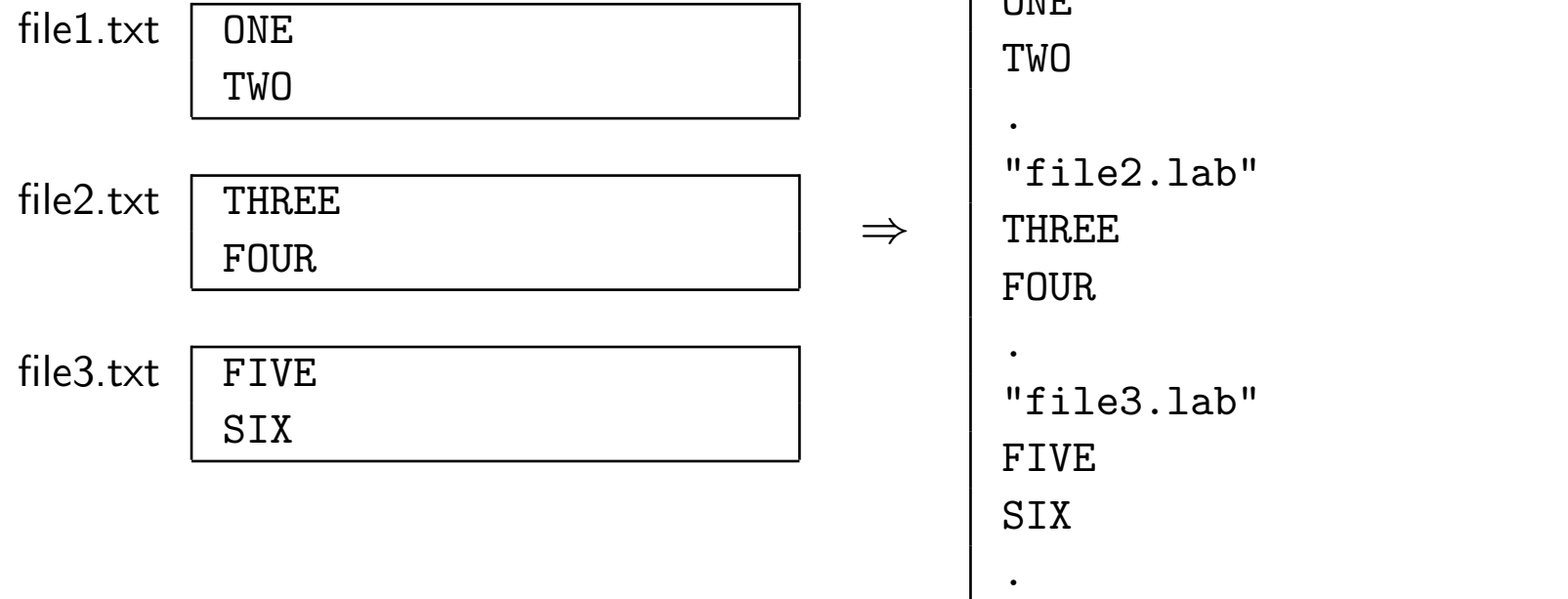
HLEd **Edit Command**

Edit commands are given in a plain text file, one command per line. Use HLEd -Q to show a list of supported commands. Example:

```
CH/C X A Y B      - replace Y in context of A_B by X
DC/X A B C ...    - define context A as labels B,C,...
DE/D A B ...      - delete labels A,B,...
DL [N]            - delete all current level [or level N]
EX/E              - expand labels from dictionary or form A_B_C
FG/I X            - fill interlabel gaps with label X
FI/F A Y B        - find all occurrences of pattern AYB
IS A B            - insert A at start and B at end
IT/Z              - ignore triphone contexts in CH/FI cmds
LC/L [X]          - convert phonemes to Left-context dependent
ML N              - move to level N (1-99)
ME/M X A B ...    - merge label seq A,B,.. and rename X
NB/V X            - label X is not an interword boundary
...
```

Merging multiple label/MLF files (HLEd)

Command: HLEd -i <OUTPUT_MLF> /dev/null <INPUT_LABEL1> <INPUT_LABEL2> ...



Can also use MLF files as input, hence merging multiple MLF files.

Extracting MLF subset (HLEd)

Command: HLEd -i <OUTPUT_MLF> -I <INPUT_MLF> /dev/null file1.lab file3.lab

```
#!MLF!#  
"file1.lab"  
ONE  
TWO  
.  
"file2.lab"  
THREE  
FOUR  
.  
"file3.lab"  
FIVE  
SIX  
.
```

⇒

```
#!MLF!#  
"file1.lab"  
ONE  
TWO  
.  
"file3.lab"  
FIVE  
SIX  
.
```

Can also use the '-S' option to specify the subset of logical files.

Converting Word to Phone Transcriptions (HLEd)

Command: HLEd -d <DICTIONARY> -i <OUTPUT_MLF> <EDIT_FILE> <INPUT_MLF>

Dictionary:

ONE	w ah n
TWO	t uw

Edit file:

EX	# expand pronunciations
IS sil sil	# Insert start/end labels

```
#!MLF!#
"/sentence.lab"
ONE
TWO
.
```

⇒

```
#!MLF!#
"/sentence.lab"
sil
w
ah
n
t
uw
sil
.
```

Acoustic Model Definition

HTK defines various basic components that make up the entire HMM system as **macros**. This provides a simple way to tie different components together. Macros are identified by ~ (tilde) followed by a single character denoting the macro type.

Examples:

Macro	Meaning
~o	An option macro
~h	A hidden Markov model
~s	An HMM state
~t	An HMM state transition matrix

There are many other macro types for Gaussian components, mean vectors, covariance matrices, adaptation transformation matrices, etc.

Option Macro Definition

Option macro specifies the basic parameterisation for the HMM system:

```
~o
<STREAMINFO> 1 39
<VECSIZE> 39<NULLD><MFCC_E_D_A><DIAGC>
```

- <STREAMINFO> – specifies the stream widths
 - the above example defines a single stream with 39 dimension
 - can also split 39 into 3 equal streams:
<STREAMINFO> 3 13 13 13
- <VECSIZE> – specifies the expected feature vector size
- <NULLD> – specifies the duration model type (NULLD = no duration modelling)
- <MFCC_E_D_A> – specifies the parameter kind
- <DIAGC> – specifies the covariance matrix kind (DIAGC = diagonal covariance matrix)

HMM State Macro Definition – Single Gaussian Component

HMM State macro specifies the parameters for an HMM state:

```
~s "macro_name"  
<MEAN> 39  
0.0 0.0 0.0 ..  
<VARIANCE> 39  
1.0 1.0 1.0 ...  
<GCONST> ...
```

- <MEAN> – specifies the Gaussian mean vector
- <VARIANCE> – specifies the Gaussian variance vector (only the diagonal elements needs to be stored)
- <GCONST> – specifies the constant value independent of observation vector

HMM State Macro Definition – Gaussian Mixture Model (GMM)

```
~s "macro_name"  
<NUMMIXES> 2  
<MIXTURE> 1 0.4  
<MEAN> 39  
0.0 0.0 0.0 ..  
<VARIANCE> 39  
1.0 1.0 1.0 ...  
<GCONST> ...  
<MIXTURE> 2 0.6  
<MEAN> 39  
0.0 0.0 0.0 ..  
<VARIANCE> 39  
1.0 1.0 1.0 ...  
<GCONST> ...
```

- <NUMMIXES> – specifies the no. of mixture components

HMM Transition Matrix Macro Definition

```
~t "macro_name"  
<TRANSP> 5  
1.0 0.0 0.0 0.0 0.0  
0.0 0.5 0.5 0.0 0.0  
0.0 0.0 0.5 0.5 0.0  
0.0 0.0 0.0 0.5 0.5  
0.0 0.0 0.0 0.0 0.0
```

- <TRANSP>
 - first specifies the number of states N (including non-emitting start/end states)
 - followed by $N \times N$ values of transition probabilities
 - row – **from** states
 - column – **to** states

HMM Macro Definition

Put state and transition matrix macros together (inline or as macros):

```

~h "macro_name"
<BEGINHMM>
<NUMSTATES> 3
<STATE> 2
<MEAN> 39
0.0 0.0 0.0 ...
<VARIANCE> 39
1.0 1.0 1.0 ...
<GCONST> ...
<TRANSP> 3
1.0 0.0 0.0
0.0 0.5 0.5
0.0 0.0 0.0
<ENDHMM>

```

or

```

~s "state"
<MEAN> 39
0.0 0.0 0.0 ...
<VARIANCE> 39
1.0 1.0 1.0 ...
<GCONST> ...

```

```

~t "transition"
<TRANSP> 3
1.0 0.0 0.0
0.0 0.5 0.5
0.0 0.0 0.0

```

```

~h "macro_name"
<BEGINHMM>
<NUMSTATES> 3
<STATE> 2
~s "state"
~t "transition"
<ENDHMM>

```

Master Macro File (MMF)

Each HMM definition can be saved in separate files. Alternatively, all macros can be put together in one single file. This file is called the **Master Macro File (MMF)**. It is also possible to store macros in multiple MMF files. The HMM macros are stored one after another.

```
~o
<STREAMINFO> 1 39
<VECSIZE> 39<NULLD><MFCC_E_D_A><DIAGC>
~h "hmm1"
<BEGINHMM>
...
<ENDHMM>
~h "hmm2"
<BEGINHMM>
...
<ENDHMM>
...
```

- MMF stores an inventory of HMM definitions
- The set of models defined by a **model list**
 - plain text with HMM macro names per line
- Can also define **logical** models:

```
model1
model2  model1
model3  model1
model4
```

- model1 and model4 are physical models
- model2 and model3 are logical models

HMM Prototype Construction

An HMM prototype is needed for HMM model initialisation (manually created). An example of a 3-state HMM prototype with left-to-right topology:

```
~o
<STREAMINFO> 1 39
<VECSIZE> 39<NULLD><MFCC_E_D_A><DIAGC>
~h "prototype"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<MEAN> 39
0.0 0.0 0.0 ...
<VARIANCE> 39
1.0 1.0 1.0 ...
<GCONST> ...
```

```
<STATE> 3
...
<STATE> 4
...
<TRANSP> 5
1.0 0.0 0.0 0.0 0.0
0.0 0.5 0.5 0.0 0.0
0.0 0.0 0.5 0.5 0.0
0.0 0.0 0.0 0.5 0.5
0.0 0.0 0.0 0.0 0.0
<ENDHMM>
```

Note: Macro name must match file name. Hence, above file needs to be named “prototype”.

HMM Training – Flat Start (HCompV)

HMM training using Baum-Welch algorithm is an iterative process. Need an initial model:

- Port an existing model from other domain/task (if available)
- Do a **flat start** (all models in the system has the same parameters)

How to determine the parameters for the initial model?

- Use global mean as the Gaussian mean
- Use global variance as the Gaussian variance

This can be achieved using HCompV:

Command: `HCompV -M <HMMDIR> -f 0.01 -m -S <TRAIN_DATA_SCRIPT> <PROTOTYPE>`

- by default, HCompV will update the variances
- '-m' – update mean of the prototype with the global mean
- '-f x ' – generate a variance floor vector set at x times global var.

Clone HMM's from Prototype

Updated prototype:

```
~o
<STREAMINFO> 1 39
<VECSIZE> 39<NULLD><MFCC_E_D_A><DIAGC>
~h "prototype"
<BEGINHMM>
...
<ENDHMM>
```

Variance floor macro:

```
~v "varFloor1"
<VARIANCE> 39
0.01 0.01 0.01 ...
```

⇒

Flat start MMF:

```
~o
<STREAMINFO> 1 39
<VECSIZE> 39<NULLD><MFCC_E_D_A><DIAGC>
~v "varFloor1"
<VARIANCE> 39
0.01 0.01 0.01 ...
~h "hmm1"
<BEGINHMM>
...
<ENDHMM>
~h "hmm2"
<BEGINHMM>
...
<ENDHMM>
...
```

HMM Training – Baum Welch Re-estimation (HERest)

HERest performs *embedded* re-estimation:

- HERest -T 1 -H <SRC> -M <TGT> -I <TRANS> -S <TRAIN_SCRIPT> <MODEL_LIST>

Usually between 4 to 8 iterations are performed. Setting '-T 1' enables basic tracing:

```
Processing Data: data1.mfcc; Label data1.lab
Utterance prob per frame = -8.912119e+01
Processing Data: data2.mfcc; data2.lab
Utterance prob per frame = -8.677007e+01
...
Reestimation complete - average log prob per frame = -8.587275e+01
  - total frames seen           = 1.124823e+06
```

This gives you:

- Average log probability per frame per utterance
- Average log probability per frame for all the training data
- Total number of frames seen (how much data used for training)

Parallellising Model Training using (HERest)

To cope with large amount of training data (hundreds or thousands of hours), it is necessary to break the training process into smaller tasks so that they can be run on multiple computers.

To achieve this, HTK supports a 2-stage training process:

- Accumulation of sufficient statistics
- Combining multiple sets of sufficient statistics and perform parameter estimation

Stage 1:

- `HERest -p <ID> -H <SRC> -M <TGT> -I <TRANS> -S <TRAIN_SCRIPT> <MODEL_LIST>`
 - This will generate the sufficient statistics in `<TGTDIR>/HER<ID>.acc`

Stage 2:

- `HERest -p 0 -H <SRC> -M <TGT> <MODEL_LIST> <TGT>/HER1.acc <TGT>/HER2.acc ...`

Iterative Mixing-up for GMM Building (HHEd)

Usual practice is to begin with a single Gaussian component per HMM state. To increase the number of Gaussian components, use HHEd as follows

```
HHEd -H <SRC> -M <TGT> <EDIT_COMMANDS> <MODEL_LIST>
```

Example, the following edit command increases the number of Gaussian component to 2 for state 2 to 4 of all the HMM's:

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

Alternatively, can also specify relative increment. The following example increases the number of Gaussian component per HMM state by 2:

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

- HTK recursively split the heaviest component (largest component prior) until desired number of components achieved.
- Usually split $2 \rightarrow 4 \rightarrow 8 \rightarrow 16$
- After every split, perform 4–8 Baum-Welch iteration to refine model parameters.

HMM Testing (HVite)

HVite is used to perform Viterbi decoding from a decoding network. To perform decoding, the following items are needed:

- acoustic model (e.g. MMF file & model list)
- decoding network (generated by HParse or HBuild – see previous lecture)
- Vocabulary (pronunciation dictionary)

The command for running Viterbi decoding is:

```
HVite -H <MMF> -i <OUTPUT_MLF> -S <TEST_SCRIPT> <DICTIONARY> <MODEL_LIST>
```

Options for HVite

- '-t x ' – set pruning threshold to x
- '-v x ' – set word end pruning threshold to x
- '-s x ' – set grammar scale factor to x
- '-p x ' – set word insertion penalty to x

HMM Evaluation (HResults)

The recognition output (MLF) can be compared with the reference (MLF) to obtain word error rate performance:

```
HResults -I <REFERENCE> <MODE_LIST> <OUTPUT>
```

Example output:

```
===== HTK Results Analysis =====
Date: Mon Feb 16 16:28:27 2009
Ref : referemce.mlf
Rec : output.mlf
----- Overall Results -----
SENT: %Correct=30.00 [H=30, S=70, N=100]
WORD: %Corr=80.00, Acc=66.67 [H=1200, D=100, S=200, I=200, N=1500]
=====
```

- $\text{Corr} = \frac{H}{N} \times 100\%$; $\text{Acc} = \frac{N-S-D-I}{N} \times 100\% = \frac{H-I}{N} \times 100\%$
- $\text{WER} = 1 - \text{Acc} = \frac{S+D+I}{N} \times 100\%$

Recap

- Introduction to HTK
- Data collection and preparation
- MFCC feature extraction
- HMM initialisation
- Training HMM models
- Iterative Mixing-up for GMM building
- Testing using HMM models
- Performance Evaluation