# Automatic Speech Recognition 2008-09: Lab-session (3) — Continuous speech recognition (part 1 of 3) —

**Hiroshi Shimodaira** (*Revision* : 1.1)

In this tutorial you will

- train monophone models on the Resource Management (RM) database using HTK[1].
- investigate how recognition accuracy changes with more training
- vary the amount of pruning during recognition (and possibly during training)
- vary the language model scaling
- plot accuracy against the various parameters

and in parts 2 and 3 you will go on to try more sophisticated HMMs (e.g. multiple-mixture models, tied-mixture models, context-dependent models), and different language models.

# 1 Preparation

## 1.1 Initialisation

```
% cd ~/asr
% mkdir ASR
% cd ASR
% ~hshimoda/pub-asr/bin/init-t3
```

You will find following directories under `~/asr/ASR/`

| dir names | contents |
|---|---|
| file_lists/ | lists of files used for training and recognition |
| labels/ | phone and word labels |
| model_lists/ | list of model names (= monophones) used in this experiment |
| recognised/ | recognition results |
| resources/ | HTK configuration files, dictionaries, and language models |
| scripts/ | script files for training, recognition, and summarising results |

The online manual of HTK is available at this url:

`~/asr/ASR/manuals/htkbook/index.html`

## 1.2 Data

Data is from the DARPA Resource Management Continuous Speech Corpora (RM) provided by LDC

`http://www.ldc.upenn.edu/Catalog/LDC93S3B.html`

You don't need to parameterise it, it's already converted to MFCCs and stored under

`/home/hshimoda/pub-asr/corpora/RM`

Do not copy the data either, use the central copy (the scripts are set up to do this).

There are about 3000 training utterances and 1200 test ones (in 4 sets of 300). The state-of-the art results on this data are word error rates (WER) of about 5% on each of the first 3 sets and 9% on the Sept 92 set. This is using the same word-pair language model provided for this practical. You won't get quite this level of accuracy, mainly because we are going to use monophone models rather than context-dependent models, but you can get close. See also Rabiner and Juang section 8.8.

The master copy of RM, which includes speech wave files, can be found at

`/group/corpora/public/resource_management/rm2`

[1] http://htk.eng.cam.ac.uk

### 1.3 Computing resources

Training takes about $15 \sim 20$ seconds CPU time per iteration on the Linux machines with the scripts as provided.

You would want to run a "bigger" job which takes much longer CPU time later on. In such case, try to avoid using machines where other user jobs are already running.

Useful commands to check how many people are logged into a machine, and how many jobs are running are "`users`" and "`top`". Use `ssh tlab3` (for example) to remote log in.

## 2 Procedures (how to run the script files)

### 2.1 Outline

A typical procedure for training phone HMMs with HTK is as follows, where typical HTK command names are shown in parentheses as well.

- P1: with a small size of speech corpus in which phone labels are available, train each phone HMM individually. (HCompV → HInit → HRest)
- P2: with the same data set, carry out "embedded training" for the set of HMMs. (HERest)
- P3: with a large size of speech corpus with sentence-level transcriptions, train the set of HMMs again. (HERest)

In this lab session, we assume P1 and P2 have been done already, and we start with P3 followed by recognition experiments.

A typical sequence of procedures for an experiment will consist of following three steps:

| | Descriptions | Provided script files (HTK commands) |
|---|---|---|
| Step-1: | Training the models | `./scripts/train_monophones` (HERest) |
| Step-2: | Recognition | `./scripts/recognise_with_monophone_models` (HVite) |
| Step-3: | Analysing the results | `./scripts/results` or<br>`./scripts/results_summary` (HResults) |

Note that the script files should be run at your ASR directory (i.e. `~/asr/ASR`) and nowhere else.

In the Step-1 and Step-2 above, there are some parameters you can change to investigate the effect to the performance of training or recognition.

### 2.2 Step-1: training the models

Training is iterative. Each successive iteration of models is stored in a different sub-directory of "`models/R1/`". if you use the provided script file, you will start with training the initial models which have been already stored in "`models/R1/hmm0`", and output (trained models) is stored into a file under "`models/R1/hmm1`" . In next iteration, you will use models in "`models/R1/hmm1`" and result is stored into "`models/R1/hmm2`", and so on. Initially, there are only a few those directories under the `models/R1` directory. Other directories `models/R1/hmm`$N$, where $N > 2$, should be created by yourself before you continue further training iterations.

The initial models provided in "`models/R1/hmm0`" have been trained on the TIMIT corpus.

you will have to find out how many iterations of training are required.

Read the comments in the training script.

Anyway, try the following example to get idea

```
% cd ~/asr/ASR
% ls -l models/R1                          ··· you will see several directories there
% ls -l models/R1/hmm0                     ··· initial model is in "MODELS"
% less  models/R1/hmm0/MODELS              ··· you can view the models
% ls -l models/R1/hmm1                     ··· no files there yet!
% ./scripts/train_monophones 0             ··· start training "hmm0"
% ls -l models/R1/hmm1                     ··· you will now find a file there !
```

### 2.3 Step-2: recognition

Just try to run the script provided

```
% ./scripts/recognise_with_monophone_models 1
% ls -l recognised/R1            ··· you will find recognition output files have been created
```

The first argument "1" tells the script to use models in "`models/R1/hmm1`" to recognise the test data of RM.

Recognition speed varies depending on the parameters. Try turning pruning off altogether...

#### 2.3.1 Language model

The language model provided is a simple word-pair grammar, in HTK format (i.e. a finite state graph). For each word, the probabilities of all allowed following words are equal. You will have a chance to try your own unigram or bigram language models later on.

### 2.4 Step-3: analysing results

Run the script to get results for each of the four test sets

```
% ./scripts/results 1
```

The first argument "1" means the iteration number again.
To get overall results, use `results_summary`.
Make sure you understand the output from `HResults`.

**BEWARE:** If you have too much pruning, then the decoder will fail on some files. `HResults` will only report results on actual recogniser output – so make sure it reports 1200 sentences (or at least, not *too many* fewer than that), otherwise you might get misleading WER figures.

## 3 Experiments

### 3.1 things to investigate

1. Plot WER against at least those parameters:
   - pruning level
   - language model scaling factor
   - number of training iterations

   whose details are described in following sections.
2. Measure speed

### 3.2 Parameters for recognition

`HVite` has several parameters you might want to change. Plot graphs of WER against the value of each parameter (holding other parameters fixed). Note that the effects of the parameters interact.

You might want to write scripts to automate this process, and collate the results automatically. If you have no idea about Shell script programming, visit this web page for a quick introduction:

http://www.faqs.org/docs/Linux-HOWTO/Bash-Prog-Intro-HOWTO.html

and feel free to ask me if you need help with writing shell scripts.

Note that the provided script files are not editable (i.e. write protected). you need to create copies of them with different file names from the original ones so that you can edit your own script files to change parameters. The following example will create your own script '`my_train_monophones`" against the original "`train_monophones`" :

```
% cd scripts
% cp -p train_monophones  my_train_monophones
% cd ..
```

- **Pruning**: (`-t` flag to `HVite`, start with values around 90, say 50 to 120) Stick to the same HMMs (a fully trained set, not iteration 0). Alter the script to only use one of the four test sets if things are running slowly (but do not plot results for one test set on the same plot as for another, or all four). The scripts are provided with beam width set to 70, which probably is too harsh but makes the recogniser run fast.
- **Language model scaling factor**: (`-s` flag to `HVite`). When the log probabilities for acoustic model and language model are summed, the LM can be weighted. The scripts use a reasonable default, try varying the value either side of this. This parameter will need re-tuning when you try a different language model later on.
- **Word insertion probability**: (`-p` flag to `HVite`). Controls insertion/deletion ratio. A good rule of thumb is to make number of insertion and deletion errors equal. I would optimise this parameter after setting the LM scaling factor.

## 3.3 Parameters for training

During training, you can vary the amount of $\alpha$ and $\beta$ pruning by changing the argument `-t 100 100 600`, where the first 100 is a log probability beam width. If training fails at that level of pruning (i.e. all $\alpha$s and $\beta$s get pruned), HERest will add another 100 (the second number after the `-t`) and try that utterance again. It will keep adding 100 until it reaches 600, then give up. **Leave this until later**: you can return to this section and try tightening up that pruning to see if good models are still trained. For now, just train some models up using the current parameters. To understand what $\alpha$ and $\beta$ pruning is, you need to understand the derivation of the Baum-Welch algorithm.

## 3.4 Measuring speed

You will have noticed that some scripts preface commands with `time`. Unsurprisingly, this reports the time a program takes to run (try `time ls`). You should record the "user" time and **not** the "real" time. The latter is the actual elapsed time which will be longer than the time the process actually spent running on the CPU, since UNIX machines run many processes at once by allowing them to take turns on the CPU. On a heavily loaded machine, the "real" time may be considerably more than the "user" time. The "system" time is the time spent waiting for things like file access and should generally be small. Note that the format of the output from `time` may vary slightly across machines running different versions of Unix.

**BEWARE:** `HVite` is actually run on each of the four test sets individually, so make sure you scroll back up the terminal and add up all four times, or pipe standard error to a file and examine it afterwards:

```
% ./scripts/recognise_with_monophone_models 1  >& stderr.file
% grep user stderr.file
```

**IMPORTANT:** to make speed comparisons, always measure the speed on *the exact same type of machine*. Note that a couple of the machines in the lab (at the front) are slightly different hardware to the rest.

## 3.5 Hints

- Train a set of models and do several recognition experiments *which means you don't need to retrain the models each time*. Then retrain the models with different pruning, and do the same recognition experiments.
- Try and keep the machines busy - you can even do testing on one model set whilst training another, to save time, just don't get in a muddle. And, as ever, keep a clear record of your experiments and results.

# 4 Assignment

You should complete all 3 parts (part 2 and 3 will be available next week) of this assignment and submit a lab report.

## Lab report

Write a lab report – keep the length around 10 pages (summarise results using tables and graphs, do not include lengthy program output listings). Include graphs of WER vs. pruning, WER vs. speed, etc. etc.

Analyse how each parameter you varied affected performance. Which combination gave the lowest WER? Why you got such a result? Which parameter settings gave a good compromise between WER and speed (use graphs to show this)? Comment on the performance of different language models (the word pair one supplied, a unigram built by you and at least one bigram built by you).

The report is due Thursday, 26th, March, 2009 at 4pm; use the Informatics submit command to submit your report in PDF format.

**What's expected:** Before you carry out experiments and write a report, it is important that you get familiar with "scientific method" and "scientific report".

http://en.wikipedia.org/wiki/Scientific_method
http://www.unc.edu/depts/wcweb/handouts/lab_report_complete.html

Your report is supposed to follow the scientific method/report.

You should complete at least the main part of the practical

- for acoustic models
  - train single-mixture monophone models for several iterations
  - try different numbers of components in the Gaussian mixture distributions of monophone models
  - make plots of WER against various parameters
  - try single-mixture and tied-mixture word-internal triphone models
- for language models
  - evaluate one of your LMs through an ASR experiment. (acoustic models can be chosen arbitrary)

For a higher mark, you need to attempt some or all of:

- find out how speed and accuracy trade off
- try more sophisticated triphone models, e.g. state-clustered triphone models
- try LM experiments given in part 3
- try more than one language models in ASR
- omit the delta-delta, and possibly also the delta, coefficients from the observations and see what happens to the WER