Automatic Speech Recognition 2011-12: Lab-session (3)
— Continuous speech recognition (part 1 of 3) —

Hiroshi Shimodaira (*Revision* : 1.1)

In this lab session you will

- train monophone models on the WSJCAM0 database using HTK[1]
- investigate how recognition accuracy changes with more training
- try different amount of pruning during recognition (and possibly during training)
- try different values of the language model scaling
- plot accuracy against the various parameters

1 Preparation

1.1 Initialisation

You should use the following work directory ('*WorkDir*' here after) that is allocated to you in this course throughout the experiments.

`/afs/inf.ed.ac.uk/group/teaching/asr/Work/` *YourLoginName*

where *YourLoginName* denotes your login name.

```
% cd WorkDir
% /afs/inf.ed.ac.uk/group/teaching/asr/bin/init-t3
```

You will find the following directories under *WorkDir*.

dir names	contents
corpus/	original corpus (wsjcam0) and parameterised data (WSJCAM0)
file_lists/	lists of files for training and recognition
labels/	phone and word labels
model_lists/	list of HMM model names
dictionaries/	word lists and pronunciation dictionaries
language_models/	language models
scripts/	script files for training, recognition, and summarising
configs/	HTK configuration files
edfiles/	HTK edit command files
logs/	log files
recognised/	recognition output
manuals/	copy of the HTK book, reference papers

The online manual of HTK is available at this url:
WorkDir`/manuals/htkbook/index.html`

HINT If you want to avoid typing the long path name of *WorkDir*, it would be a good idea to create a symbolic somewhere under your home directory. For example, if you've already had `~/asr` , the following will create a symbolic link, `~/asr/ASR`, whose path name can be used instead of *WorkDir*.

```
% cd ~/asr
% ln -s WorkDir ASR
```

1.2 Data

Data is from the WSJCAM0 Cambridge Read News corpus provided by LDC

`http://www.ldc.upenn.edu/Catalog/LDC95S24.html`

You don't need to parameterise it, as it's already converted to MFCCs, and those MFCC files can be found under

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

`corpora/WSJCAM0`

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

There are about 8000 training utterances (from 90 speakers), 700 development utterances (from 20 speakers), and 20000 test ones (from 20 speakers).

The master copy of WSJCAM0, which includes speech wave files, can be found at under corpora/wsjcam0, which is a symbolic link to

`/group/corpora/public/wsjcam0`

The corpus is divided into sub sets for experiments as follows:

training set	si_tr
development sets	si_dt5a, si_dt5b, si_dt20a, si_dt20b
evaluation sets	si_et5a, si_et5b, si_et20a, si_et20b

where "5" and "20" denote a data set with vocabulary of about 5000 and about 20000, respectively.

For details, please see those PDF files under the manual directory, `manuals`

"fransen-pye:wsjcam0:1994.pdf"	corpus description
"robinson-fransen:icassp:1995.pdf"	evaluation

2 Procedures

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/word-level transcriptions, train the set of HMMs again. (HERest)

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

	Descriptions	Provided script files (HTK commands)
Step 0:	Preparation of speech data dictionary language model	
Step 1:	Model training	
(a)	model initialisation	`./scripts/prep_monophones` (HCompV)
(b)	initial training	`./scripts/init_monophones` (HInit,HRest)
(c)		`./scripts/merge_monophones`
(d)	embedded training	`./scripts/train_monophones_wo_sp` (HERest)
(e)	silence model refinement	`./scripts/mk_sp_model`
(f)	label realignment	`./scripts/align_mlf` (HVite)
(g)	embedded training	`./scripts/train_monophones` (HERest)
Step 2:	Recognition	`./scripts/recognise_with_monophones` (HVite)
Step 3:	Result analysis	`./scripts/show_results` (HResult)

Note that the script files should be run at your *WorkDir*directory and nowhere else.

Step 0 has been done already. Step 1 and Step 2 are the main focus of this lab session. In Step 1 and Step 2, there are some parameters you can change to investigate the effect to the performance of training or recognition.

For details on HTK commands, please refer to the HTK Book, whose copy is available in the manual directory.

2.1 Step 1: Model training

2.1.1 Model preparation

First of all, make sure that you are in the right directory, i.e. the top directory of this project (*WorkDir*).

```
% cd WorkDir
```

It should be noted that all the script commands shown hereafter are supposed to be run in this directory, otherwise they will not work properly.

Initial phone models (HMMs) have been stored in `proto/`, where you will find 45 files, each of which corresponds to a phone HMM.

Since HMMs are stored in ascii format at this stage, you can take a look at them. Try

```
% less proto/zh
```

you will see that models parameters such as mean vectors, variances of each state and transition probabilities have been set to default initial values. For details on HMM definition files, please refer to HTK Book section 7.2.

Now, run the following script, which calls HCompV to initialise model parameters based on training data.

```
% ./scripts/prep_monophones
```

The output is saved in directory `models/R1/hmmP`. Take a look at a file in the directory to see which parameters have been updated by the command.

2.1.2 Initial training

Run this command:

```
% ./scripts/init_monophones                    ··· this would take more than 10 minutes
```

This script runs HInit and HRest to train each phone model separately by referring phone labels given in a Mater Label File (MLF), `labels/phone/si_tr.mlf` . New models are saved in `models/R1/hmmI` (by HInit) and `models/R1/hmmR` (by HRest).

Detailed logs of the commands are saved in directory, `logs/init_monophones` .

2.1.3 Embedded training

So far, each phone HMM has been stored in a separate file. Hereafter, all the model files are saved into a single file for efficiency.

Run the following script

```
% ./scripts/merge_monophones
```

which will create a merged model file, `models/R1/hmm0/MODELS` .

Run the following script to carry out "*embedded training*" (HERest) iteratively

```
% ./scripts/train_monophones_wo_sp 0 1 2 3
```

where the arguments `0 1 2 3` denote model numbers to train, i.e. it first trains `hmm0` and stores the output into `hmm1`, which is then used as an input for next training to produce `hmm2`, and so on. As a result, the newest model will be `hmm4`.

2.1.4 Silence model refinement

Run this command to create another silence model, *sp*.

```
% ./scripts/mk_sp_model 4
```

which will store a new model set in `hmm5`. For details, please see HTK Book section 3.2.2.

Retrain the models by the following command

```
% ./scripts/train_monophones 5 6
```

which calls HERest twice, and the newest models will be now `hmm7` .

2.1.5 Label realignment

Now, it's time to create new phone labels using the current models. Run this command

```
% ./scripts/align_mlf 7
```

which will crate a new MLF, `labels/phone/mono-algned2.mlf` .
(This will take a couple of minutes.)

2.1.6 Model retraining with new labels

Run the following command to retrain the models.

```
% ./scripts/train_monophones_realigned 7 8
```

In the above example, we repeat embedded training two times. However, the optimal number of iteration varies depending on the data you use. You could investigate how the number of training iteration would affect recognition accuracy on test data.

2.2 Step 2: Recognition

Assuming the newest model set you got is `hmm9`, run the following command

```
% ./scripts/recognise_with_monophones 9
```

which runs HVite on test data set ("si_dt5a") using `hmm9` in `models/R1` .

The recognition output is stored in the directory, `recognised/R1` .

```
% ls -l recognised/R1                          ··· you will get a list of recognition output files
% less recognised/R1/hmm9_si_dt5a_output.mlf
```

Recognition speed and accuracy vary depending on the parameters. Please see
Try turning pruning off altogether...

2.3 Step 3: Result analysis

Run the script to get results

```
% ./scripts/show_results 9
```

The first argument "9" denotes the model number used for recognition.

Make sure you understand the output from `HResults` (HTK Book section 3.4.1 and 17.19.1).

BEWARE: If you have too much pruning, the decoder (HVite) will fail on some files and give an error message of "No token survived ...", and then `HResults` will not give you correct statistics because it reports results only on actually recognised output.

– so make sure it reports 368 sentences (for "si_dt5a" set), i.e. $N = 368$, 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 200) 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 α and β 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 αs and βs 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 α and β 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 `/usr/bin/time`. Unsurprisingly, this reports the time a program takes to run (try `time ls`), which would be shown in the last lines in the log file.

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.

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 of the assignment (part 2 and 3 will be available next week and the week after next). Your coursework submission is complete only if you have done the following two submissions:

- submission of report (in PDF or Word format)
- submission of your ASR system that gave the highest recognition accuracy

The submission deadline is Thursday, 5th April 2012 at 16:00.

Your coursework will be assessed mainly based on

- quality of experiments/investigation
- quality of report and discussions

Note that your coursework should be done solely by yourself, meaning you cannot use reports/documents, programs, files, or experimental results of someone else (except the programs/scripts provided in the course). However, you may have discussions with your colleagues. For plagiarism, please see:

`http://www.inf.ed.ac.uk/teaching/plagiarism.html`

4.1 Submission of report

1. Download a report template package from the course web page. There are two versions available, one for LaTeX, the other for Microsoft Word (Open Office). You can use either one.
2. As you will find in the package, your report submission should consist of the following two documents:
   (a) coversheet
   (b) report

   For the purpose of anonymous assessment of reports, do not write your name or matriculation number on your report, but write your "pseudonym" on the first page of the report. A pseudonym can be any word(s) or letter/digit sequence but your own name. On your coversheet, indicate your name, matriculation name, and pseudonym.
3. Submit your report and coversheet files with the "submit" command in DICE computing environment. The following shows an example of submitting two files, "report.pdf" and "coversheet.pdf".

   ```
   % submit asr 1 report.pdf coversheet.pdf
   ```
4. You should receive an email of acknowledgement from the system as soon as your submission has been received successfully. Keep the message as an evidence of your coursework submission.

4.2 Submission of the best ASR system

This part can be minimal as long as you use *WorkDir* and you keep all the relevant files (i.e. all the models you trained, and other files needed to run the system) there. If it is not the case, please contact the course lecturer for advise.

Having all the relevant files stored in *WorkDir*, go to "scripts" directory under *WorkDir*, and create a recognition script whose file name is "`run_the_best_system`". It could be just a copy of your recognition script that gave the best result. Make sure that HMM model directory and HVite options are set properly in the script so that it gives the same result (accuracy) as the one for the best system described in your report.

4.3 Report writing

Your report should be prepared using report templates, which will be provided on the course web page shortly. Keep the length of your report between 5 and 8 pages including figures, tables, and references.

You should look into the following points and write a "scientific report". For higher marks, you need to give good discussions based on both theories and experiments, and you will also need to try the optional items.

1. Monophone models
    - investigate how the three parameters (-t, -s, -p) of HVite influence recognition performance (i.e. accuracy and speed), and summarise the results using graphs, e.g. graphs of WER vs. pruning threshold, WER vs. speed, etc.
    - find the optimal number of mixture components (optional)
    - carry out experiments using different feature vectors, e.g. MFCC without delta features (optional)
2. Tied-state triphone models
    - investigate how the number of clusters influences recognition performance by varying the threshold value for "TB" command of HHEd (see `./scripts/mk_tied_triphones`). Draw a graph of recognition accuracy in terms of the total number of free model parameters (or the number of clusters).
    - seek the optimal configuration of parameters that gives highest recognition accuracy (optional).
3. MLLR speaker adaptation
    - investigate the relationship between the number of regression classes and recognition performance.
    - try adaptation experiments for different HMMs (e.g. different numbers of mixture components, triphone models) [optional]
    - investigate the relationship between the amount of training samples and recognition performance. [optional]
    - based on the experiments, discuss the limitation of this type of adaptation and what modification could be done. [optional]

The following are some technical instructions on writing reports.

- experimental results should be efficiently summarised using figures or tables (but not both if possible) - avoid using a separate graph/table for each experiment.
- figures/tables should be numbered and captioned.
- experimental conditions and methods should be shown clearly, using a table for example.(e.g. data sets used for training and evaluation, relevant parameters used for training and evaluation) Providing sufficient information is very important for scientific reports (see below) so that other people could redo the same experiments.
- show results even if there was no improvement. It is important to discuss/analyse why there was no improvement.
- for "scientific method" and "scientific report", please see

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

Tips on experimental design

You will see that there are a large number of combinations of parameters which will affect training speed, recognition speed and recognition accuracy. Although it does not take much time for carrying out a single session of experiment with the default parameter set, it will take much longer for certain sets of parameters. In addition, it's not feasible to try all of the possible combinations of the parameters. Plan what experiments are needed and how they should be carried out.

To carry out experiments efficiently, it is a very good idea that you create script files which test various parameter combinations automatically.

You should start experiments as soon as possible in order not to miss the deadline.