

Automatic Speech Recognition 2015-16: Coursework and Assignment 1 — Continuous speech recognition —

Hiroshi Shimodaira and Steve Renals

(Revision : 1.2)

In this coursework you will

- train monophone models on the WSJCAM0 database using HTK¹
- 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
- see what effects that have on accuracy
- increase the number of Gaussian mixture components in the pdfs for the monophone models
- see what effects that has on accuracy
- convert monophone models into triphone (i.e. context dependent) models using a decision tree-based clustering technique
- convert the single Gaussian triphone HMMs into multiple mixture component triphone HMMs
- see what effects those have on accuracy

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-asr.sh
```

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/ | Shell(Bash) script files for training, recognition, and summarising |
| python/ | Python version of the above |
| 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<http://htk.eng.cam.ac.uk>

1.2 Data

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

<http://catalog.ldc.upenn.edu/LDC95S24> <https://catalog.ldc.upenn.edu/LDC95S24>

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

`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/ws_jcam0`, which is a symbolic link to

`/group/corpora/public/ws_jcam0`

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. In this coursework, we will use `si_dt5a` as a test set to evaluation the recognition performance of the final system, along with its subset `si_dt5a.div3` to run quick test.

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

“fransen-pye:ws_jcam0:1994.pdf” corpus description
“robinson-fransen:icassp:1995.pdf” evaluation

2 Monophone models

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 corpus² 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 | <code>./scripts/prep_monophones</code> (HCompV) |
| (b) | initial training | <code>./scripts/init_monophones</code> (HInit, HRest) |
| (c) | | <code>./scripts/merge_monophones</code> |
| (d) | embedded training | <code>./scripts/train_monophones_wo_sp</code> (HERest) |
| (e) | silence model refinement | <code>./scripts/mk_sp_model</code> |
| (f) | label realignment | <code>./scripts/align_mlf</code> (HVite) |
| (g) | embedded training | <code>./scripts/train_monophones</code> (HERest) |
| Step 2: | Recognition | <code>./scripts/recognise_with_monophones</code> (HVite) |
| Step 3: | Result analysis | <code>./scripts/show_results</code> (HResult) |

²In this coursework, we use the same data set for P1, P2, and P3.

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 create a new MLF, `labels/phone/mono-aligned2.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.

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 `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 – single Gaussian component monophone models

3.1 Things to investigate

1. Plot WER against at least those parameters:

- pruning level
- language model scaling factor
- word insertion probability (penalty)

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.

Examples of shell scripts relevant to this coursework are available:

</afs/inf.ed.ac.uk/group/teaching/asr/ASR/ShellScriptExamples>

See '0.README' in the directory for details.

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_recognise_with_monophones' against the original "recognise_with_monophones" :

```
% cd scripts
% cp -p recognise_with_monophones my_recognise_with_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).
- **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 α s and β 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 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 Multiple mixture component monophone models

After you have carried out the embedded training of single Gaussian monophone models several times to get `hmm9`, it's time to increase the number of mixture components of each state. This is done by iterative “mixture splitting” with the `HHEd` command.

4.1 Copying source models to a new directory

At first, you need to determine the model number to increase the mixture component. If it is “`models/R1/hmm9`”³, do as follows (here I assume your current directory is *WorkDir*):

```
% mkdir models/R2                ... we are going to use a new directory, R2.
% mkdir models/R2/hmm0            ... create a directory where source models are stored.
% cp models/R1/hmm9/MODELS models/R2/hmm0    ... copy the monophone models.
```

4.2 Splitting

Run the following script, which calls `HHEd`, to split a single Gaussian to two components.

```
% ./scripts/mixup
```

This automatically creates a new directory, `models/R2/hmm1`, for new models.

You can take a look at the new models to see what changes have been made:

```
% less ./models/R2/hmm1/MODELS
```

³Further experimental results will depend on which model you've chosen here. It would be interesting to see how different performance you will get depending on the initial model number.

4.3 Retraining

The new models above need retraining. Run the following command to run HERest for three times to get models up to hmm4.

```
% ./scripts/train_monophones_realigned -R2 1 2 3
```

The option `-R2` is essential, as it tells the script to use the models under `R2` rather than the default directory `R1`.

4.4 Evaluation

Test the new models in `models/R2/hmm4` with the following command:

```
% ./scripts/recognise_with_monophones_small -R2 4
```

which will produce recognition output:

```
recognised/R2/hmm4_si_dt5a-div3_output.mlf
```

and log files:

```
logs/recognise/R2/log-last           ... a whole log of the command invoked last
logs/recognise/R2/log-4-si_dt5a-div3 ... digest of the log, which is also shown on display
```

Make sure that you have not got the warning message of “No token survived...” many times (a few times should be fine).

Run the following command to get recognition accuracy.

```
% ./scripts/show_results_small -R2 4
```

Are the new models more accurate than the single-Gaussian component models from part 1? (It would be interesting to evaluate other previous models, e.g. `hmm1`, `hmm2`, `hmm3`.)

It should be noted that those script files with “`_small`” in name indicate that the script uses a small test set, “`file_lists/si_dt5a-div3.scp`” (123 utterances) rather than the subset of “`file_lists/si_dt5a.scp`” (368 utterances). Using a small test-set can make the evaluation much faster, but it might loose accuracy in estimating recognition performance. However, we need to carry out a large number of experiments in the lab sessions, we will use this small subset for evaluation.

4.5 Increasing more number of mixture components

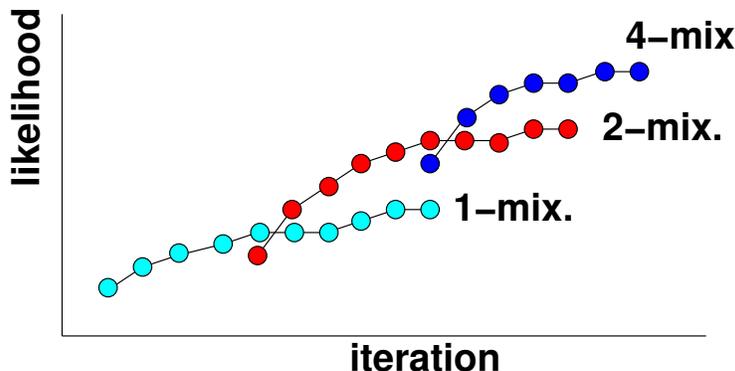
We can repeat the above splitting and retraining operation to get models with more number of mixture components.

The question is how we should change the numbers of mixture components along with repetitive training. Recall the local-optimum problem with HMM parameter estimation - it would not be a good idea that we increase the number of mixture components very quickly, e.g. $1 \rightarrow 10 \rightarrow 50$. The following table shows an example of the process:

| # of mix. components | splitting | retraining | final model |
|----------------------|---------------------------|---------------------|-------------|
| (1) | - | - | (R1/hmm9) |
| 2 | hmm0 \rightarrow hmm1 | hmm1, hmm2, hmm3 | hmm4 |
| 3 | hmm4 \rightarrow hmm11 | hmm11, hmm12, hmm13 | hmm14 |
| 4 | hmm14 \rightarrow hmm21 | hmm21, hmm22, hmm23 | hmm24 |
| 5 | hmm24 \rightarrow hmm31 | hmm31, hmm32, hmm33 | hmm34 |
| 7 | hmm34 \rightarrow hmm41 | hmm41, hmm42, hmm43 | hmm44 |
| 10 | hmm44 \rightarrow hmm51 | hmm51, hmm52, hmm53 | hmm54 |
| \vdots | \vdots | \vdots | \vdots |

Try increasing the number of mixture components further to see how recognition accuracy/speed changes, and draw graphs⁴. To that end, read the next section 4.6 at first, and copy “scripts/mixup” to your own script⁵, e.g. “scripts/my-mixup”, and edit the script to change some definitions. You will also need to copy “edfiles/mixup2.hed” to your own edit command file, e.g. “edfiles/my-mixup.hed”.

Go over your notes and make sure you understand what you just did, what mixtures of Gaussian are, and why increasing the number of mixture components improves accuracy.



4.6 Details on mixup and HHed commands

The “mixup” script calls HHed in the following manner:

```
HHed -C configs/config.basic -T 1 \
-H models/R2/hmm0/MODELS \
-M models/R2/hmm1 \
edfile/mixup2.hed \
model_lists/mono-sp.list
```

Here models/R2/hmm0/MODELS is used as source models, and a new model file is stored in the directory models/R2/hmm1. Operation (edit) commands such as increasing the number of mixture components should be given to HHed through a file, which is specified as the 1st argument to HHed.

As you will see, the edit command file, “edfile/mixup2.hed”, consists of only one line:

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

where “2” followed by “MU” is the number of Gaussian mixture components you want to have for each state (i.e. 2, 3, and 4) of all the models.

5 Decision-tree-based tied-state triphone models

We can improve the monophone models by making them context dependent. Among several types of training algorithms for context-dependent models, we will employ a decision tree-based clustering technique to have triphone models whose parameters are effectively shared, i.e. “tied”.

The procedure will look like this

Step 1a create triphone labels and triphone model list.

Step 1b create single-Gaussian cross-word triphones by cloning single-Gaussian monophone models.

Step 1c train single mixture cross-word triphones.

Step 2a carry out clustering to create single-Gaussian tied-state triphone models from the models above.

Step 2b train the tied-state triphone models.

Step 3 split Gaussian component to get multiple-mixture tied-state triphone models, and train and evaluate the models.

⁴It is a good idea that you compare the graph of likelihood on training data and the graph of recognition accuracies on test data.

⁵You will find another script, scripts/mixups, which is a more sophisticated version of the script, and does not require you to prepare the mixup.hed instruction file.

5.1 Step 1: Cross-word triphone models

5.1.1 Step 1a: preparing triphone labels

Run the following script to convert phone labels into context dependent form and also create a list of triphone models that appear in the training data.

```
% ./scripts/mk_xwrd_labels
```

Make sure that a context dependent (triphone) label file, `labels/phone/xwrd_tri.mlf` has been created from `labels/phone/mono-aligned2.mlf` and a list of triphone models, `model_lists/xwrd_tri.list` has been created as well.

5.1.2 Step 1b: creating cross-word triphone models

Run the following script to create initial cross-word triphones by simply cloning monophone models in `models/R1/hmm9`.

```
% ./scripts/mk_xwrd_triphones
```

The initial model will be stored under `models/R7/hmm0`.

Because the total file size of triphone models is much larger than that of monophone models, model files hear-after will be stored in binary format rather than ascii format for efficiency, and thus you cannot take a look at them directly.

5.1.3 Step 1c: training cross-word triphone models

Run the following command to train the triphones up to `hmm2` (or further).

```
% ./scripts/train_xwrd_triphones 0 1 2
```

The cross-word triphone models created so far are only those triphones that appeared in the training set, and thus a lot of triphones might be missing.

5.2 Step 2: Single-Gaussian tied-state triphone models

5.2.1 Step 2a: clustering HMM states

Run the following script to carry out top-down clustering of HMM states

```
% ./scripts/mk_tied_triphones
```

which will produce a long message both on display and to a file:

```
% less logs/train/R9/log-hhed
```

You will find a line like this:

```
TB: Stats 24->1 [4.2%] { 60309->2086 [3.5%] total }
```

which shows that a total of 60,309 HMM states were merged into 2,086 clustered.

The script above creates those lists:

| | |
|--|---|
| <code>model_lists/full_tri.list</code> | list of all possible triphone models |
| <code>model_lists/trees</code> | current questions defined and trees |
| <code>model_lists/treeg.list</code> | physical models and associated logical models |

See HTK Book section 3.3.2, 10.5, and 17.8 for details, and see the script file, `scripts/mk_tied_triphones` to understand how HHed is called for clustering.

5.2.2 Step 2b: training tied-states triphone models

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

5.2.3 Step 2c: evaluation

Run the following scripts.

```
% ./scripts/recognise_with_tied_triphones_small 4
% ./scripts/show_results_small -R9 4
```

5.3 Step 3: Mixup – multiple mixture triphone models

Much the same way for the monophone models,

```
% ./scripts/mixup_tied
```

which will read in `models/R9/hmm4` and store the new models with two mixture components into `models/R9/hmm10`.

```
% ./scripts/train_tied_triphones 10 11 12 13
```

Copy `mixup_tied` to your own file to change the number of mixture components and model numbers, and repeat the procedure above.

For evaluation, the same evaluation scripts shown in Step 2c can be used.

6 Assignment

Your coursework submission is complete only if you have done the following three submissions:

- electronic submission of report (in PDF or Word format) via “submit” command
- submission of a hard copy of your report to ITO
- submission of your ASR system that gave the highest recognition accuracy for the test-set `si_dt5a` (not `si_dt5a_div3`)⁶.

The submission deadline is Wednesday, 24th February 2016 at 16:00.

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/misconduct, please see:

<http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct>

There is also a policy on late coursework:

<http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests>

6.1 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.

⁶You can use the subset `si_dt5a_div3` for trying different parameter combinations and models, but you should also report the performance of the best system using `si_dt5a`.

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.
5. Submit a hard copy of your report (coversheet + report) to ITO.

6.2 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.

6.3 Report writing and marking guidelines

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 4 and 7 pages including figures, tables, and references. Experimental results should be well summarised using figures or tables.

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.

1. Monophone models
 - Design of experiments and presentation of results [20 marks]
 - Investigate how the three parameters (-t, -s, -p) of HVite influence recognition performance [7 marks]
 - Find the optimal number of mixture components [7 marks]
 - Additional experiments (e.g. carry out experiments using different feature vectors, e.g. MFCC without delta features) [6 marks]
2. Tied-state triphone models
 - Design of experiments and presentation of results [20 marks]
 - Investigate how the number of clusters influences recognition performance [10 marks]
 - Seek the optimal configuration of parameters and models that gives highest recognition accuracy [10 marks]
3. Discussions and quality of report [20 marks]

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.

Appendix A. Implementation of decoding parameters in HTK

ASR system uses $P(X|W)$ and $P(W)$ to find the best word sequence W .

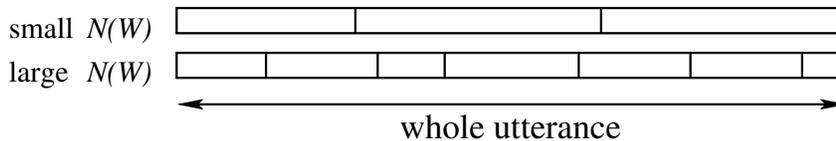
- These estimated probabilities are different in reliability and dynamic range.
- More shorter words tend to have lower scores than those with fewer longer words in real implementation (due to $P(W)$).

$$\begin{aligned} W^* &= \arg \max_W P(X|W) P(W) \\ &\Rightarrow \arg \max_W P(X|W) P(W)^{LMS} IP^{N(W)} \quad \dots \text{modified formula} \\ &= \arg \max_W \log P(W|X) + LMS \log P(W) + N(W) \log IP \end{aligned}$$

LMS : language model weight (LM scaling factor)

IP : insertion penalty

$N(W)$: number of words in W



- Interpretation of LMS
As $LMS \rightarrow 0$, $P(W)^{LMS} \rightarrow 1$,
i.e. the smaller LMS becomes, the less important the LM is.
- Interpretation of IP
Assuming a uniform LM (every word has an equal occurrence probability), $P(W)$ for fewer longer words (i.e. smaller $N(W)$) is greater than $P(W)$ for more shorter words (i.e. larger $N(W)$). IP is used to balance this.
 - $0 < IP \leq 1$ (i.e. $\log IP < 0$) the smaller IP becomes, the more shorter words are penalised (i.e. fewer longer words are preferred)
 - $1 < IP$ (i.e. $\log IP > 0$) the larger IP becomes, the more shorter words are preferred. (i.e. the more insertion errors.)
- LMS and IP are determined heuristically (on validation data). Larger LMS usually needs larger IP .