Machine Learning and Spoken Language Technology

MLSALT11: Speech and Machine Learning Practical

Speech Practical: Large Vocabulary Speech Recognition

1 Introduction

This practical examines three parts of a large vocabulary speech recognition: language modelling; acoustic model speaker adaptation; and system combination. All the approaches will be examined on systems built for a Broadcast News transcription task. This practical handout is aimed to be self contained. However the relevant handouts from the lectures (MLSALT2 and MLSALT5) may be useful.

The practical is concerned with large vocabulary speech recognition, for these experiments Broadcast News transcription. The configuration investigated is related to those used in the CU-HTK 2003 Broadcast News Evaluation Systems [2], but updated in include models based on deep learning. Decision-tree state-clustered triphone acoustic models, trigram language models and lattices are provided. The aim of the practical is to investigate the effects of improved language and acoustic models on large vocabulary speech recognition. Software and scripts for lattice rescoring are provided. However, you will be expected to write code for language model interpolation, scoring, and system combination. This code may be written in any language.

The remainder of this document is organised as follows. The next section (section 2) describes the detailed procedure for the practical, followed by section 3, which gives some information about the practical write-up.

The theory section and underlying software tools and examples (can be used for reference if you don't want to read it now) are then given. Section 4, briefly details some of the theory required for this practical. This theory will also be covered in the lecture notes. Section 5 gives an introduction to the HTK Hidden Markov Model Toolkit, including brief descriptions of lattice rescoring tools. Section 6 introduces the HLM toolkit for manipulating language models and commands required in this practical.

The complexity of the experiments, time to run and implementation, increases as the practical progresses. You need to work consistently throughout the practical period to finish all experiments.

---

2 Practical

This practical allows you to make use of a range of speech recognition approaches. In a similar fashion to real system development, some approaches that should work don't. When this occurs (and you are confident that your code/scripts/command-line are correct), you should discuss the results, along with any ideas of why it happened.

You also only have limited time/compute resources. Performing a sweep of all possible configurations and parameters is not possible. Simply getting the best performance on the task will not guarantee the best mark [1].

Some of the steps (for example rescoring lattices with acoustic models) can take a few minutes to run. Think of the best ways to make use of this time (for example planning experiments/checking scripts/collating results). You also have a maximum of jobs that you can run on the machine stack. Planning is useful!

Additional reading and discussion is always useful. For example you may want to look at statistical significance between results.

2.1 Important Dates

The approximate timescale for the practical is (see web-pages for exact dates)

Michaelmas Week 5	Start of Practical
Michaelmas Week 8	Deadline to complete experiments in section 2.4
Lent Week 5	Challenge dev-set release
Lent Week 8 (start)	Challenge eval-set release
Lent Week 8 (end): 3:00pm	End of Practical (eval submission)
Easter (start of Full Term): 4:00pm	Submission of practical report

Some of the systems that can be built will take relatively large amounts of memory. You should be careful not to exceed your disk quota on the MPhil system. This can be avoided by deleting (use `rm`) old model sets/lattices, and intermediary model sets, that you don't need anymore.

Do *not* make copies of all files for practical in your own space - only copy the files that you need[2]. For example, do not copy the lattices, acoustic models or language models.

2.2 Files and Directories

The following files are supplied for this practical. They may be found in the directory `/usr/local/teach/MLSALT11/Speech` on the system.

[1] This is one area where the practical differs from real evaluations!

[2] You can link files and directories using `ln -s name`.

(a) `lib/testlists` - the list of shows associated with each of the tasks

- `dev03sub` - a single show to allow rapid development
- `dev03` - 6 shows (3 hours of data) that should be used for development
- `eval03` - 6 shows (3 hours of data) that should **only** be used for held-out evaluations. All configurations used to recognise this test set **must** be clearly motivated from development numbers.

(b) `lattices` - a directory containing lattices for each of the test set shows. These are the lattices generated from `HDecode`, using the phonetic PLP model set in `hmms` and an interpolated bigram language model using the same data sources as those in `lms`. They are used for the rescoring experiments in this practical.

(c) `scripts` - example scripts (and script for scoring output). Including:

- `1bestlats.sh`: generate 1-best from the lattices
- `mergelats.sh`: determinise/prune a lattice
- `score.sh`: score the output in a directory
- `lmrescore.sh`: apply a new language model to the lattices
- `hmmrescore.sh`: rescore lattices with a new hmm
- `hmmadapt.sh`: generate a set of CMLLR transforms
- `cnrescore.sh`: generate confusion networks

(d) `hmms` - contains a set of HMMs. There are six acoustic models available:

1. `plp`: baseline HMM-GMM acoustic models built using PLP features and the phonetic lexicon;
2. `tandem`: HMM-GMM acoustic models built using tandem features and the phonetic lexicon;
3. `hybrid`: HMM-DNN acoustic models built using PLP features and the phonetic lexicon;
4. `grph-plp`: baseline HMM-GMM acoustic models built using PLP features and the graphemic lexicon;
5. `grph-tandem`: HMM-GMM acoustic models built using tandem features and the graphemic lexicon;
6. `grph-hybrid`: HMM-DNN acoustic models built using PLP features and the graphemic lexicon;

The files are:

- `MMF.`*system*: is the HTK Master Model File (MMF)
- `xwrd.clustered.`*system*: the list of all logical and physical models associated with the MMF
- `stats`: the state occupancy counts (used for regression class tree generation) for the phonetic systems;

- `stats-grph`: the state occupancy counts (used for regression class tree generation) for the graphemic systems.

The set of HMMs used for this experiment are state-clustered cross-word triphone models. There are approximately 700 distinct states (including the silence models) with 16 Gaussian components per state for the speech models and 32 Gaussian components per state for the silence models. These models were trained using Minimum Phone Error (MPE) estimation.

(e) `lib/flists` - a directory containing lists of the segmented development and evaluation data for each of the shows. Thus the set of segments for show `20010117+2000+2100+PRI+TWD` is in

- `20010117+2000+2100+PRI+TWD.scp`

(f) `lms` - the set of trigrams (and the interpolated bigram) language models. There are five trigram language models available, see Appendix C for a description of each of the data used to build each of the models. In addition there is the word-list file for the lms in `lib/wlists`.

(g) `lib/cfgs` - configuration files for the experiment. Including

- `mllr.cfg` - `MLLRMEAN` transform config
- `cmllr.cfg` - `CMLLR` transform config
- `hlm.cfg` - an example configuration file for the HLM tools `LPlex` and `LMerge`

(h) `lib/texts` - contains the correct transcriptions for the `dev03` test sets mapped into a form for stream probability generation. The files are

- `dev03.dat`

(i) `lib/dicts` - a directory containing two set of dictionaries (or lexicons).

1. `phonetic` - the words are split into phones, all these dictionary file names start with `train`;
2. `graphemic` - the words are split into phones, all these dictionaries file names start with `train-grph`;

For each of the types of dictionary there are three files. Taking as the example the phonetic lexicon, there are:

- `train.dct`: base dictionary;
- `train.hv.dct`: dictionary where silence models `sil` and `sp` have been added to the pronunciations. This dictionary is used with `HVite`;
- `train.lv.dct`: dictionary where pronunciations for sentence start `<s>` and sentence end `</s>`. This dictionary is used with `HDecode`.

For more information about the dictionaries see appendix D.

(j) `lib/trees` - a directory containing the trees for mapping the confidence scores. There is a different tree for each of the six acoustic models.

(k) `lib/wlists` - a directory containing the word-lists associated with the language models.

(k) `base/bin` - a directory containing the binaries required for this experiment (as well as additional binaries not required). You will find the following binaries in this directory

- `HDecode`: binary used for speech recognition with the HMM-GMM systems;
- `HDecode.mkl`: binary used for speech recognition with the HMM-DNN (hybrid) systems;
- `HERest`: binary used for estimating acoustic models and adaptation transforms;
- `HLRescore`: binary for manipulating lattices;
- `HLConf`: binary for producing confusion networks
- `LPlex`: used for computing stats from a language
- `LMerge`: used for merging language models

To get started you will need to link some of these directories, and copy the `scripts` directory (so can improve/update the scripts) into the directory where you are going to run the practical:

```
ln -s /usr/local/teach/MLSALT11/Speech/lib
ln -s /usr/local/teach/MLSALT11/Speech/lms
ln -s /usr/local/teach/MLSALT11/Speech/lattices
ln -s /usr/local/teach/MLSALT11/Speech/base
ln -s /usr/local/teach/MLSALT11/Speech/hmms
cp -r /usr/local/teach/MLSALT11/Speech/scripts
```

You may find it useful to use *shell scripts* [3] to perform repeated operations and/or run experiments for this practical.

2.3 Scripts

To help in running experiments a number of scripts (see (c) above) are provided. By default all the scripts (other than the scoring script `score.sh`) will produce one (or more) `LOG` file (sometimes with extensions). This will be very useful in understanding exactly what command has been run.

Prior to using any of scripts make sure you understand what the script does and where the `LOG` file, for example, will be generated. The scripts supplied are meant to help you. For historical reasons they are a mix of `tcsh` and `bash` - you should feel free to rewrite/modify/improve.

[3] see `http://www-h.eng.cam.ac.uk/help/tpl/unix/scripts/scripts.html` for help on shell scripts

Some of the scripts are able to be run without altering them (especially for the start of each of the sections). If you simply run the script without understanding what it is doing, the practical will take *significantly* longer than if you spend some time understanding the scripts.

The command-line options for a script are obtained by running the script with no arguments. See the practical experiments for more details on examples of using the scripts.

The scripts make use of the *Sun Grid Engine* (SGE) to submit and manage jobs on the machines available for this practical. To check what jobs you currently have running use

```
qstat
```

To submit jobs you can use the `qsub` command - for examples of its use see the example scripts. You can also look at the Moodle web-page for this:

`https://www.vle.cam.ac.uk/mod/book/view.php?id=2294772`

2.4 Experiments: Language Modelling

This part of the practical aims to improve the language model. The acoustic model used to generate the lattices in this section is the baseline phonetic `PLP` model. Note the command line options for all script can be obtained by just running the script with no arguments.

The deadline to complete the experimental part of the section practical is the end of Michaelmas term. When you have completed this part of the practical you need to briefly discuss your results with the demonstrator. If you are concerned that you will not complete this section by the deadline you MUST talk to the demonstrator as soon as possible, or contact the practical organiser.

1. First obtain the 1-best output from the lattices with the bigram language model that was used to generate the lattices on the `dev03sub` development set. This can use the supplied script `1bestlats.sh`

   ```
   ./scripts/1bestlats.sh dev03_DEV001-20010117-XX2000 \
          lattices decode plp-bg
   ```

   To allow similar the same scripts to be used for various tasks and naming conventions there is a mapping used between names (see `lib/shownames/bne.convmap`). The convention is

   ```
   "testset"_"showname"
   ```

   `dev03sub` is a subset of `dev03`are the same for rapid scoring. Thus this command generates the 1-best for the show `20010117+2000+2100+PRI+TWD` which

is the only show in dev03sub. A LOG is generated by this script as well as the MLF file showing the recognition output. These are useful when trying to debug errors in your set-up. The LOG and MLF files can be found in

```
plp-bg/dev03_DEV001-20010117-XX2000/1best/LM12.0_IN-10.0/LOG
plp-bg/dev03_DEV001-20010117-XX2000/1best/LM12.0_IN-10.0/rescore.mlf
```

Look in these files and see how the command relates to the descriptions in section 5. Look in the HTK manual if you need any of the options described. The MLF file should then be scored. This make use of the supplied script score.sh.

```
./scripts/score.sh plp-bg dev03sub 1best/LM12.0_IN-10.0
```

this will generate a directory scoring. The file

```
scoring/sclite/dev03sub_plp-bg_1best_LM12.0_IN-10.0.ctm.filt.sys
```

will contain the recognition scores. Check that you get an error rate of:

15.3% [9.5% substitutions, 3.0% deletions, 2.7% insertions]

on the dev03sub development set. See Appendix E for a more detailed description.

2. Evaluate the performance in terms of Word Error Rate (WER) of the 5 supplied language models using all the dev03 shows. This can make use of the supplied script lmrescore.sh. For example to rescore show 20010117+2000+2100+PRI+TWD using the language model in lms/lm1,

```
./scripts/lmrescore.sh dev03_DEV001-20010117-XX2000 lattices decode \
  lms/lm1 plp-tglm1 FALSE
```

Again a LOG file and MLF file will be generated in

```
plp-tglm1/dev03_DEV001-20010117-XX2000/rescore/LOG
plp-tglm1/dev03_DEV001-20010117-XX2000/rescore/rescore.mlf
```

Examine how these commands relate to the descriptions in section 5. This command should be run for all the shows in dev03 (you may consider writing a script for this). Once this command has been run for *all* the shows in dev03 they may be scored using

```
./scripts/score.sh plp-tglm1 dev03 rescore
```

the results will be in

```
scoring/sclite/dev03_plp-tglm1_rescore.ctm.filt.sys
```

Is the performance as you expect given the language model data used for each of the sources?

3. Find the perplexity of each of the language models on the `dev03` development set transcriptions (using the `LPlex` command and supplied text source `lib/texts/dev03.dat`). Do the perplexities correlate well with the WER?

4. Design and implement a language model interpolation tool based on minimising the perplexity. The tools can be written in any language. It is only necessary to implement a tool for estimating the interpolation weights given the stream of probabilities from `LPlex`. See section 6 for more details. Ensure that you include the configuration file `hlm.cfg` when using these programs. Note you should only use the development data-set, `dev03`, to estimate the interpolation weights.

   Thus the recommended approach is:

   i) using `LPlex` obtain a stream of probabilities for each of the LMs to be interpolated

   ii) estimate interpolation weights using the tool that you write

   iii) merge LMs using the interpolation weights and `LMerge` (for an example use of LMerge see page 12)

   You will need to consider how to initialise the interpolation weights - is the estimate sensitive to the initialisation?

5. Evaluate the performance of the language models obtained with your interpolation tool on the `dev03` development set in terms of perplexity and WER. Are the same trends followed on the `eval03` test set?

6. The language model interpolation weights have so far been estimated on the development data-set, `dev03`. Though a sensible approach if all the test-sets are similar in nature, for example all on the same topic, it is interesting to examine what happens if show specific interpolation weights are used. As the shows in recognition are unknown, it is necessary to perform *unsupervised language model adaptation*. Here rather than fixing the language model interpolation weights, it is possible to optimise them at run-time, given initial recognition hypotheses. For each show separate interpolation weights are generated. The shows are then re-recognised using the *show-specific* language model.

   The basic procedure to perform is for *each* show in `eval03`:

   i) using the global, `dev03` tuned, interpolation weights (from (3)) generate the 1-best hypotheses;

   ii) map the 1-best hypotheses to an appropriate text data format to train the interpolation weights.(see section 6);

   iii) compute interpolation weights using the data using the mapped hypotheses in (ii), and merge the LMs;

   iv) apply the new show-specific LM to the lattices.

You should compare the performance of these show-specific LMs to the original LM in terms of both perplexity and WER on `eval03`.

2.5 Experiments: Acoustic Model Adaptation

Rather than refining the language model, it is possible to improve the acoustic model by performing *speaker adaptation*.

1. When lattice rescoring with new acoustic models, it is necessary to first determinise the lattices. To perform this on the supplied lattices

   ```
   ./scripts/mergelats.sh dev03_DEV001-20010117-XX2000 lattices decode plp-bg
   ```

   By default the merge process performs aggressive *beam* pruning and *arcs-per-second* pruning. These lattices can be rescored with the original models

   ```
   ./scripts/hmmrescore.sh dev03_DEV001-20010117-XX2000 plp-bg merge plp-bg plp
   ```

   Note this stage takes about 5 minutes to run. To monitor progress you can count the number of files that have been decoded using the log file. For example

   ```
   egrep -c File plp-bg/dev03_DEV001-20010117-XX2000/decode/LOG
   ```

   Comment on performance compared to the original lattices? Note the form of pruning that is used for `HLRescore` is different to that during the decoding stage [4].

2. Using the 1-best hypothesis generated from the bigram lattice, produce "cascaded" `CMLLR` and `MLLR` transforms. The adaptation script performs two distinct steps for this (as well as some manipulation of the MLF file):

   (a) *force align* the hypothesis using the acoustic model and the hypothesis. This generates a phone-sequence associated with the hypothesis. This uses the `HVite` HTK tool;

   (b) *transform estimation* using `HERest`. The default version produces a (global) `CMLLR` and two MLLR transforms.

   A command is supplied that performs these operations:

   ```
   ./scripts/hmmadapt.sh dev03_DEV001-20010117-XX2000 plp-bg \
           decode plp-adapt-bg plp
   ```

   Look at the files generated in

   ```
   plp-adapt-bg/dev03_DEV001-20010117-XX2000/adapt
   ```

   These transforms can then be used to rescore lattices:

[4] For a default configuration this leads to a 0.3% absolute *increase* in the WER. Do not be concerned about this, but don't forget to mention it in the report!

```
./scripts/hmmrescore.sh -ADAPT plp-adapt-bg adapt \
        dev03_DEV001-20010117-XX2000 plp-bg merge plp-adapt-bg plp
```

The last term in the above command specifies the acoustic model. There are six models supplied

- `plp`
- `grph-plp`
- `tandem`
- `grph-tandem`
- `hybrid`
- `grph-hybrid`

To get information about the other command line options, simply type the script name with no arguments. For example this script allows you to do decoding based on different lattices (as well as using adaptation transforms see below). The output can be scored using

```
./scripts/score.sh plp-adapt-bg dev03sub decode
```

3. The above process can be repeated but using the supervision from another system, cross adaptation. First the lattices can be rescored using the graphemic PLP system

```
./scripts/hmmrescore.sh dev03_DEV001-20010117-XX2000 plp-bg merge \
       grph-plp-bg grph-plp
```

Score the output from this system. Is the perform surprising? Now this hypothesis can be used for adaptation:

```
./scripts/hmmadapt.sh -OUTPASS adapt-grph-plp dev03_DEV001-20010117-XX2000 \
      grph-plp-bg decode plp-adapt-bg plp
```

These graphemic system hypothesis transforms can then be used to rescore lattices:

```
./scripts/hmmrescore.sh -ADAPT plp-adapt-bg adapt-grph-plp \
       -OUTPASS decode-grph-plp \
       dev03_DEV001-20010117-XX2000 plp-bg merge plp-adapt-bg plp
```

Again score the output from this system and comment on the performance. Note the transforms that are generated are *specific* to the model that was used to generate. If you want to use adapted graphemic systems new transforms need to be generated.

2.6 Experiments: System Combination

The above experiments have used Viterbi decoding and the hypothesis from a single system. It is also possible to combine the hypotheses from multiple systems.

1. The first stage is to write a tool that allows you to take two MLF files and compute the error rate between them. Using the two MLF files generates in the previous section:

   ```
   plp-bg/dev03_DEV001-20010117-XX2000/decode/rescore.mlf
   grph-plp-bg/dev03_DEV001-20010117-XX2000/decode/rescore.mlf
   ```

   You can compare the alignment and error rates that you obtain with the ones from the standard scoring procedure from HTK, `HResults`. To generate the `HResults` output, first convert one of the MLF files into a form suited for scoring (you can check the HTK book for what this does):

   ```
   ./base/bin/HLEd -i plp-bg/dev03_DEV001-20010117-XX2000/decode/score.mlf \
        -l '*'  /dev/null \
        plp-bg/dev03_DEV001-20010117-XX2000/decode/rescore.mlf
   ```

   Then using this "reference" score the output from the graphemic system:

   ```
   ./base/bin/HResults -t -f \
      -I plp-bg/dev03_DEV001-20010117-XX2000/decode/score.mlf \
      lib/wlists/train.lst \
      grp-plp-bg/dev03_DEV001-20010117-XX2000/decode/rescore.mlf
   ```

   From the HTK book the settings of insertion, deletion and substitutions costs are:

substitution	10
insertion	7
deletion	7

2. It is possible to perform idealised combination. Here a network is generated and the best path (sometimes referred to as the `oracle` error rate) is used for scoring. Modify the MLF combination scheme to combine two (or more hypotheses) and generate an MLF that allows the oracle error rate to be found. An example of the format of the generated MLF to combine three hypotheses is:

   ```
   #!MLF!#
   "*/DEV001-20010117-XX2000-en_FFWXXXX_0068025_0069727.rec"
   1500000 6000000 BRIAN 0.86
   6000000 11200000 STARTED_<ALTSTART>_<DEL>_<ALTEND> 0.99_0.20
   11200000 13100000 HIS 1.00
   13100000 19100000 MILLY_<ALTSTART>_MOLLY_<ALT>_MILITARY_<ALTEND> 0.97_0.99_1.00
   19100000 23900000 CAREER 0.93
   23900000 25100000 AS 0.84
   .
   ```

This means that STARTED has been aligned with !NULL so is optionally deletable, and MILLY, MOLLY and MILITARY were aligned together.

To simplify this "ideal combination" the filter used in scoring (than generates alternatives such as hasn't and has not) needs to be disabled. This means that you will need to generate new baseline numbers for the individual systems. For example to score the output in section 2.5 question 2 the following command would be used:

```
./scripts/score.sh -NOFILT plp-adapt-bg dev03sub decode
```

You need to make it clear in your write-up that you have used the -NOFILT option for this section. You should also comment on the impact that using this flag has. If you use -NOFILT for other sections, remember to mention this in the description of the experiment.

Combine the hypotheses from the previous sections (scored with the -NOFILT flag) and discuss the impact on performance. Note you may want to consider additional information that is available in the MLF when aligning: the time-stamp information; and the scores. Note the scores associated with the MLFs using Viterbi decoding are likelihoods for that word, not confidence scores.

When generating the MLF, the time-stamp information should be associated with the model you are aligning to. If there is an insertion the time-stamps should be from the non !NULL token.

3. Using the previously generated decoding lattices, generate confusion networks and confidence scores. To do this, the following command can be used:

   ```
   ./scripts/cnrescore.sh dev03_DEV001-20010117-XX2000 plp-bg decode plp-bg
   ```

   This generates confusion networks and an MLF in the directory

   ```
   plp-bg/dev03_DEV001-20010117-XX2000/decode_cn
   ```

   which can then be scored as before [5]. The confusion networks are stored in the directory

   ```
   plp-bg/dev03_DEV001-20010117-XX2000/decode_cn/lattices
   ```

   and the MLF file in

   ```
   plp-bg/dev03_DEV001-20010117-XX2000/decode_cn/rescore.mlf
   ```

   The final entry in the MLF is now the confidence score. Score this new MLF, is the results as expected?

   The default confidence scores are typically too high. To address this problem the confidence scores can be mapped to better reflect the probability of the

[5] Due to a bug in the current implementation of CN generation the time-stamp information for the MLF are incorrect. Thus the tool aligns the final MLF by default.

word being correct. Confidence mapping trees for the six systems are provided (rescoring bigram lattices for `dev03sub`). To assess how this alter the performance they can be applied during scoring (note this will overwrite your previous scoring output):

```
./scripts/score.sh -CONFTREE lib/trees/plp-bg_decode_cn.tree \
      plp-bg dev03sub decode_cn
```

What is the impact of mapping the confidence scores?

4. It is now possible to combine two sets of hypotheses. Apply confusion network decoding to the graphemic system output to yield a two sets of hypotheses which have confidence scores. Modify the alignment process so that system combines the two word sequences and generates an output MLF. You will need to be able to assign a confidence score to the `!NULL` hypothesis during combination. What is the performance of the system? How does it change with the `!NULL` confidence score. Discuss any issues with this form of combination. If you want to use `mapped` confidence scores for combination you can use the following `perl` script

   ```
   base/conftools/smoothtree-mlf.pl <mapping-tree-name> <mlf-name>
   ```

   This maps the confidence scores for the MLF, which can then be "piped" into a file for scoring/combination.

5. An alternative option for system combination is to align and combine confusion networks together, called confusion network combination (CNC). The format for these confusion networks is described in appendix B. Modify the ROVER style combination to support CNC. You will need to consider distance measures for aligning confusion networks together, as well as how you want to generate the final output.

6. Check the performance of the combination schemes on the whole of the `dev03` set. You may wish to do some additional tuning, to ensure that you parameter settings. Once you have a final version, evaluate the system performance on `eval03`.

2.7 Experiments: Evaluation System Development

You have so far evaluated three distinct aspects of speech recognition systems: language models; acoustic model adaptation; and system combination. Currently these have been evaluated using PLP systems trained with either a phonetic or graphemic lexicon. There is now the opportunity to develop a final "evaluation" system. You can:

1. combine the various approaches that you have examined together (in whatever order you want);

2. make use of acoustic models that have been trained using either DNN features (`tandem` and `grph-tandem`) or a hybrid system (`hybrid`). Note that no adaptation is available for the hybrid systems.

When using different forms of acoustic model, you should **not** assume that the insertion penalties and language model scale factors are appropriate for the new acoustic models [6]. You should also consider the impact of the lattice sizes that you are using.

Remember you should do all your development on the `dev03` data, and only do the final evaluation on `eval03`. This will mimic the process that you can perform on the Challenge data (you will not have access to any references for scoring the Challenge evaluation data) during Lent term.

3 Write-Up

Your report for this practical should include the following four sections

- **Language Model Improvements**: This section should include:
  1. a description of your implementation of the LM interpolation weight estimation scheme. A listing of your code should be included as an appendix in the write-up (but does not contribute to the word-count);
  2. a description of the experiments performed to examine the impact of the language model on perplexity and WER;
  3. a description of how you ran the unsupervised adaptation of the language model and how it impacted perplexity and word error rate. You should discuss in detail the advantages and limitations of this form of language model adaptation.
- **Acoustic Model Adaptation**: This section should include:
  1. a description of the experiments performed to examine the impact of acoustic model adaptation and configurations investigated;
  2. a discussion of the cross-adaptation experiments run.
- **System Combination**: This section should include:
  1. a description of your implementation of ROVER and CNC combination. This should include a description of the alignment process and cost functions used. A listing of your code should be included as an appendix in the write-up (but does not contribute to the word-count);
  2. a description of the experiments performed to examine the impact system combination.
- **Evaluation System Development**: discuss the process that you have used to develop the evaluation system. Don't forget to include "negative" results

[6] Typically the language model scale factor is increased when the size of the feature vector increases.

that guided your design choice. You should also discuss any changes made to your evaluation system design for the Challenge data.

This section should finish with a discussion of the impact of the limitation that the practical infrastructure has imposed, both in terms of acoustic and language models, compute resources, and the lattice rescoring framework. Note for a recent evaluation system built using HTK at CUED see [3].

In total your report should be around 4,000 to 5,000 words long. You should include the location of the code that you have written for language model interpolation and system combination (and make sure that they are readable!).

References

[1] S.J. Young, G. Evermann, M.J.F. Gales, T. Hain, D. Kershaw, X. Liu, J.J. Odell, D.G. Ollason, D. Povey V. Valtchev & P.C. Woodland *The HTK Book for Version 3.4*. Cambridge University Engineering Department.

[2] M.J.F. Gales, D.Y. Kim, P.C. Woodland, H.Y. Chan, D. Mrva, R. Sinha & S.E. Tranter, "Progress in the CU-HTK Broadcast News Transcription System," in *IEEE Transactions on Audio Speech and Language Processing*, 2006.

[3] P.C. Woodland, X. Liu, Y. Qian, C. Zhang, M.J.F. Gales, P.Karanasou, P. Lanchantin & L. Wang "Cambridge University Transcription Systems for Multi-Genre Broadcast Challenge," in *IEEE ASRU Workshop*, 2015.

[4] J.G. Fiscus, "A post-processing system to yield reduced word error rates: Recogniser output voting error reduction (ROVER)," *Proc. IEEE ASRU Workshop*, pp. 347–352, 1997.

[5] G. Evermann and P.C. Woodland, "Posterior probability decoding, confidence estimation and system combination," in *Proceedings Speech Transcription Workshop*, College Park, MD, 2000.

[6] L. Mangu, E. Brill, and A. Stolke, "Finding consensus among words: Lattice-based word error minimization," in *Proceedings Eurospeech'99*, 1999.

4 Theory

This section briefly reviews some of the theory required for this practical. The first part describes the underlying theory for language modelling and language model interpolation. This is followed by a discussion of acoustic model adaptation, and finally system combination. A brief description of lattices and their use is then given, followed by a discussion of confusion networks and confidence scores.

4.1 Language Modelling and Interpolation

Language modelling is an essential part of any speech recognition system. Language models give the prior probability of any word sequence[7] $\mathbf{w} = w_0, \ldots, w_{K+1}$, $P(\mathbf{w})$. It is impractical to compute (or store) the probability for all possible word sequences. N-gram language models are the standard approximation to making the calculation of the word sequence probability practicable. For a general N-gram language model

$$P(\mathbf{w}) = \prod_{i=1}^{K+1} P(w_i|, w_1 \ldots, w_{i-1}) \approx \prod_{i=1}^{K+1} P(w_i|w_{i-N+1}, \ldots, w_{i-1}) \tag{1}$$

Thus the probability of a word occurring is dependent on the history of $N-1$ words only. A trigram language ($N = 3$) uses

$$P(\mathbf{w}) \approx \prod_{i=1}^{K+1} P(w_i|w_{i-2}, w_{i-1}) \tag{2}$$

Even when a limited history is used, for large vocabulary speech recognition systems there will be many sequences that are not observed, or are rarely seen. Maximum likelihood probability estimates based on these counts will not be reliable. Thus discounting and back-off schemes are commonly used. Furthermore to control the size of the language modelling pruning, often based on perplexity, is applied.

The performance of language models is often assessed in terms of perplexity. This may be expressed as

$$PP = (P(\mathbf{w}))^{-\frac{1}{K+1}} \tag{3}$$

For many tasks, such as Broadcast News transcription, language model training data from a wide-range of sources are used. For example, the transcriptions from the acoustic model training data and newspaper texts may be used. Some of these sources, the acoustic training data transcriptions for example, will be a good model of the word sequences that need to be recognised, whereas the newspaper texts may not be very well matched to spoken, rather than written, English. To take this into account a set of individual language models (here trigrams) may be constructed, for example one on the acoustic models, $P_{\mathtt{a}}(w_i|w_{i-2}, w_{i-1})$, and one from the newspaper text, $P_{\mathtt{n}}(w_i|w_{i-2}, w_{i-1})$. These two language models can then be interpolated

[7] Sentence start and sentence end markers are often added to the word sequences. These are assumed to be in word w_0 and w_{K+1} for a K length word string. As all words start with sentence start this is not included in the perplexity calculation.

together to yield the final language model.

$$P(\mathbf{w}) \approx \prod_{i=1}^{K+1} \left(\lambda_{\mathtt{a}} P_{\mathtt{a}}(w_i|w_{i-2}, w_{i-1}) + \lambda_{\mathtt{n}} P_{\mathtt{n}}(w_i|w_{i-2}, w_{i-1})\right) \tag{4}$$

where $\lambda_{\mathtt{a}} + \lambda_{\mathtt{n}} = 1$. The interpolation weights will reflect both how appropriate the language is for the target domain and how well the language model is trained (effectively how much training data is available).

The interpolation weights are normally estimated to minimise the perplexity on a small amount of data from the target domain. These weights may be estimated in a similar fashion to estimating the component priors (or weights in HTK notation) of the GMMs associated with the HMM states, Expectation Maximisation (EM). Let $P(\mathtt{n}|\mathbf{w}, i, \tau)$ be the probability that the news language model generated the i^{th} word w_i (this is the equivalent of $L_{jm}(t)$ for acoustic models) at the τ^{th} iteration. Similarly $P(\mathtt{a}|\mathbf{w}, i, \tau)$ is the probability for acoustic training data language model. Using Bayes' rule

$$P(\mathtt{a}|\mathbf{w}, i, \tau) = \frac{\lambda_{\mathtt{a}}^{(\tau)} P_{\mathtt{a}}(w_i|w_{i-2}, w_{i-1})}{\lambda_{\mathtt{a}}^{(\tau)} P_{\mathtt{a}}(w_i|w_{i-2}, w_{i-1}) + \lambda_{\mathtt{n}}^{(\tau)} P_{\mathtt{n}}(w_i|w_{i-2}, w_{i-1})} \tag{5}$$

Using EM the $\tau + 1^{th}$ estimate of the interpolation is then given by

$$\lambda_{\mathtt{a}}^{(\tau+1)} = \frac{1}{K+1} \sum_{i=1}^{K+1} P(\mathtt{a}|\mathbf{w}, i, \tau) \tag{6}$$

Each iteration of training is guaranteed not to increase the perplexity.

4.2 Speaker Clustering and Adaptation

Another important stage in many state-of-the-art speech recognition system is adaptation to a particular speaker, or acoustic environment. For some tasks such as Broadcast News transcription, the data is presented as a single stream of audio data. In these cases it is necessary to first *segment* the data into homogeneous blocks, i.e. blocks that contain data from a single speaker in a single environmental condition. These blocks of data must then be *clustered* together so that all the data from a single speaker in a particular acoustic environment are all in a single cluster[8]. For the speaker adaptation experiments in this practical the speaker clustering is given [9].

Once the segments have been clustered together linear transforms, or sets of linear transforms, need to be estimated for each cluster. There are three standard forms of linear adaptation schemes for Gaussian Mixture Model systems that may be used (here the naming convention used in the HTK book is adopted):

1. `MLLRMEAN`: linear transformation of the mean. This has the general form

$$\hat{\boldsymbol{\mu}} = \mathbf{A}\boldsymbol{\mu} + \mathbf{b} \tag{7}$$

[8] In practice multiple speakers may occur in the same cluster to ensure that there is sufficient data to robustly estimate adaptation transforms.

[9] The filenames in the file-lists contain this information.

This was the original implementation of MLLR.

2. `MLLRCOV`: linear transformation of the covariance matrix. This has the general form

$$\hat{\mathbf{\Sigma}} = \mathbf{A}\mathbf{\Sigma}\mathbf{A}' \tag{8}$$

This can be efficiently implemented as a transformation of the features and the means.

$$\mathcal{N}(\mathbf{o}; \boldsymbol{\mu}, \mathbf{A}\mathbf{\Sigma}\mathbf{A}') = |\mathbf{H}|\mathcal{N}(\mathbf{H}\mathbf{o}; \mathbf{H}\boldsymbol{\mu}, \mathbf{\Sigma}) \tag{9}$$

where $\mathbf{H} = \mathbf{A}^{-1}$

3. `CMLLR`: constrained MLLR transform. The transformations of the mean and the variances are the constrained to be the same. This has the form

$$\hat{\boldsymbol{\mu}} = \mathbf{H}\boldsymbol{\mu} + \tilde{\mathbf{b}}; \quad \hat{\mathbf{\Sigma}} = \mathbf{H}\mathbf{\Sigma}\mathbf{H}' \tag{10}$$

This can be efficiently implemented as a transformation of the features

$$\mathcal{N}(\mathbf{o}; \mathbf{H}\boldsymbol{\mu} + \tilde{\mathbf{b}}, \mathbf{H}\mathbf{\Sigma}\mathbf{H}') = |\mathbf{A}|\mathcal{N}(\mathbf{A}\mathbf{o} + \mathbf{b}; \boldsymbol{\mu}, \mathbf{\Sigma}) \tag{11}$$

where $\mathbf{A} = \mathbf{H}^{-1}$ and $\mathbf{b} = -\mathbf{A}\tilde{\mathbf{b}}$

For further details of how these transformations are estimated see the HTK book.

4.3 System Combination

There are a range of schemes that can be used to combine systems together. Four standard approaches are:

1. **log-likelihood combination**: log-likelihoods from different acoustic models are treated as features for a *log-linear* model;
2. **lattice rescoring**: rescore lattices generated with one systems, using a second system [10];
3. **cross-system adaptation**: the hypothesis output from one acoustic model, is used as the adaptation supervision for a second system, or the lattices for rescoring;
4. **hypothesis combination**: the hypotheses from multiple systems are combined together. Two common approaches for performing this are based on the 1-best output with confidence scores (ROVER combination) [4], or Confusion Networks (CNs) [5] where confusion networks, section 4.5, are combined

Log-likelihood system combination will not be investigated in this practical, but the other schemes will be examined.

Initially consider ROVER system combination for two systems [11]. The 1-best (most likely) outputs from the two systems, along with the confidence scores, are given in figure 1.

[10] This can be both beneficial and a hindrance. For example if improved acoustic models are used the "best" path for that model may not be in the lattice.

[11] Time-stamp information can also be included in the output, and used to help in the alignment stage

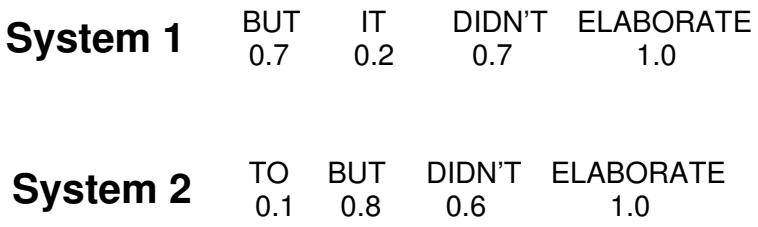

Figure 1: System Outputs

System 1		BUT	IT	DIDN'T	ELABORATE
System 2	TO	BUT		DIDN'T	ELABORATE

Figure 2: System Alignments

The first step for ROVER combination is to align the two outputs. This is a Viterbi-based approach that aims to obtain the alignment that minimises the *edit-distance* between the two word sequences. This is the same process that is used to score ASR outputs to yield WERs. For the output in figure 1, the alignment is shown in figure 2.

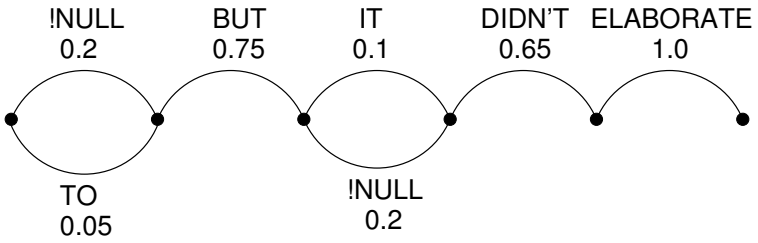

Figure 3: Word transition Network

A *word transition network* is then generated based on the alignment. This converts the alignment in figure 2 to the network in figure 3. To handle the fact that the alignment may include "deletions" and "insertions" a `!NULL` symbol is included, the probability of this arc is usually manually set. For this network the probability was set to 0.2. The scores associated with each word is the average over the two systems (other options are available).

Having generated the word transition network the best word sequence is simply obtained from the linear network by considering each set of aligned words and selecting the one with the highest score. The output from this process would then be

```
BUT DIDN'T ELABORATE
```

A third system can be simply added by aligning a new hypothesis (and confidence scores) with the current word transition network. Note this iterative process does depend on the order that the hypotheses are presented to the system.

CNC is similar to ROVER combination, however now confusion networks are aligned and combined together. There are a range of choices for alignment costs, and

combinations, between CNs. There is an opportunity to think of (and implement) options during this practical.

4.4 Lattices

Large vocabulary speech recognition is computationally expensive. Trigram and higher-order language models combined with cross-word context-dependent result in a highly complex search-space. Increasing the order of the language model, or the span of the acoustic models, can rapidly become impractical. One option to deal with this is to generate *lattices* with a simpler system, and then apply more complex models to these lattices.

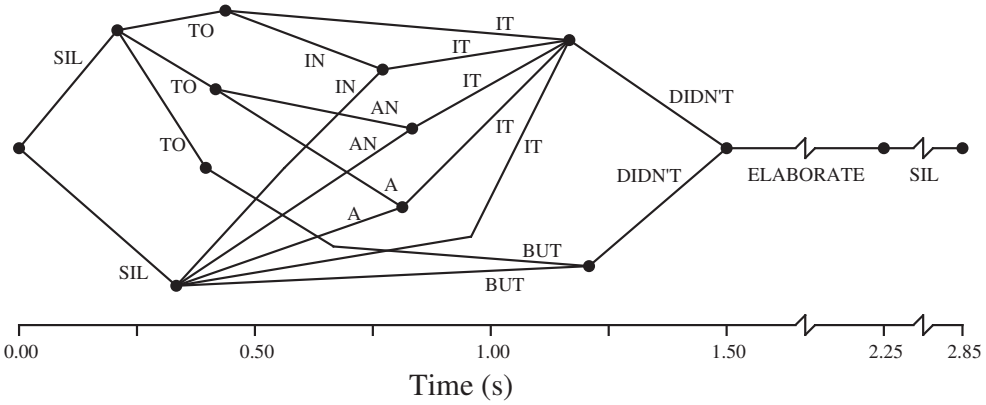

Figure 4: Example Lattice

An example lattice generated with a bigram language model is shown in Figure 4, the text format is in Appendix A. A lattice comprises:

1. a set of nodes that correspond to points in time (or word-ends);
2. a set of arcs that are labelled with word identifiers. In addition the following information can be associated with each arc:
   - acoustic score associated with that (phone-context-dependent) word;
   - language model score (taking into account the preceding word context.

The acoustic and language model scores in the lattice will correspond to the acoustic and language models used in lattice generation. There are a number of uses for these lattices, for example confidence score estimation and decoding parameter tuning. In this practical language model rescoring will be performed with lattices.

A new language model, for example a trigram, can be applied to a lattice generated by a typically simpler language model for example a bigram. The application of a trigram, to the bigram generated lattice in Figure 4 is shown in Figure 5. As the same acoustic models are being used these acoustic scores will not change. Thus it is unnecessary to read, or compute likelihoods from, an acoustic model. Furthermore only a subset of possible paths, those in the lattice, need to have language model scores computed for them (note additional arcs and nodes may be added to reflect

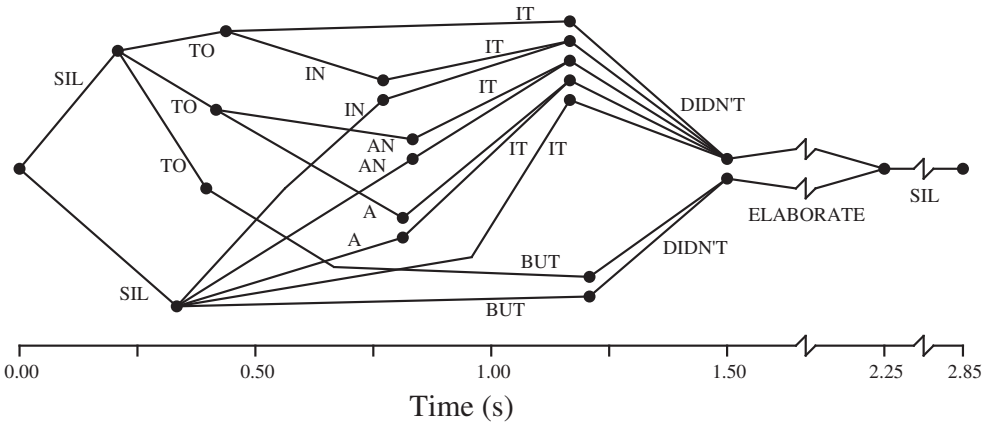

Figure 5: Example Lattice

the additional history being used in the language model). The 1-best output can then be simply generated by searching through this lattice combining the new language model scores with the existing acoustic model scores. Both of these attributes dramatically reduce the decoding time.

Though the use of lattices improves the decoding speed, there is a down-side. Since only a subset of paths are contained within a lattice, there is chance that the correct path is not contained within the lattice. In this case, however good the new language (or acoustic) model is, the correct path can never be recovered. The *lattice accuracy* is thus useful to determine what the upper-bound in performance that is achievable from rescoring the lattices. When generating lattices there is an interesting balance. Large lattices will have a lower lattice error rate, but will be slower to rescore.

4.5 Confusion Networks and Confidence Scores

The standard decoding process used in speech recognition is based on Viterbi decoding. For an observation sequence $\mathbf{O} = \{\mathbf{o}_1, \ldots, \mathbf{o}_T\}$ this yields

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \{P(\mathbf{w}|\mathbf{O}; \boldsymbol{\lambda})\} = \arg\max_{\mathbf{w}} \{p(\mathbf{O}|\mathbf{w}; \boldsymbol{\lambda}) P(\mathbf{w})\} \tag{12}$$

where $\boldsymbol{\lambda}$ are the model parameters. This estimated word sequence is the most likely sentence for the observation sequence. Though a sensible choice, depending on the task of interest other decoding options may be considered. A general expression that can be used to represent a range of (possibly task specific) decoding schemes is based on minimising the expected Bayes' risk. This decoding process can be expressed as

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left\{ \sum_{\tilde{\mathbf{w}}} P(\tilde{\mathbf{w}}|\mathbf{O}; \boldsymbol{\lambda}) \mathcal{L}(\tilde{\mathbf{w}}, \mathbf{w}) \right\} \tag{13}$$

where $\mathcal{L}(\tilde{\mathbf{w}}, \mathbf{w})$ is the "loss" between the word sequence $\tilde{\mathbf{w}}$ and the "reference" $\mathbf{w}$. In the same fashion as discriminative training this loss can be defined at various levels, for example:

- **sentence** level: the loss function here is

$$\mathcal{L}(\tilde{\mathbf{w}}, \mathbf{w}) = \begin{cases} 1, & \tilde{\mathbf{w}} \neq \mathbf{w} \\ 0, & \tilde{\mathbf{w}} = \mathbf{w} \end{cases} \tag{14}$$

  It is simple to show that this yields the standard Viterbi decoding scheme;

- **word** level: the loss function is the edit distance between word sequences;

- **phone** level: first the phone sequences for each of the word sequences is generated, and then the loss function is the edit distance between phone sequences.

For many tasks, and this practical, the criterion used to assess the system is WER. It would therefore be natural to use the word level loss function to obtain the word sequence that minimises the expected WER. One problem with examining this directly is that it is computationally expensive to consider all possible word sequences, both for the "reference" and for the posteriors. To address this problem *confusion networks* [6] are often used as an approximation to the full word-level loss function decoding. Here a set of heuristics are defined to map a lattice into a linear network, similar to the word transition networks used for ROVER system combination.

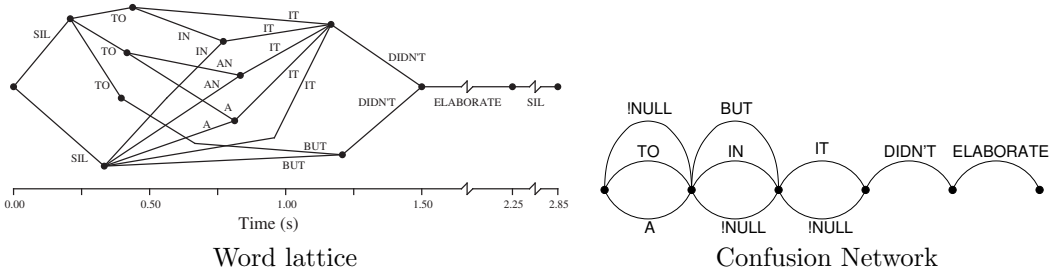

Figure 6: Example Confusion Network

An example of converting a lattice into a confusion network is shown in figure 6. Note there are also scores associated with the two forms (these scores have been omitted from the diagram):

- **word lattice**: the score for the arc comprises the language model probability and acoustic score. These are often stored separately;

- **confusion network**: the score is the posterior probability of the word arc.

Note for the confusion network (in the same fashion as the word transition networks), a `!NULL` arc is used. Decoding with a CN is the same as when using ROVER. For each each set of words in the CN, the word with the highest posterior is selected.

Another use for CNs is to extract confidence scores. A confidence score can be viewed as the probability that the hypothesised word is correct. This is a very useful score to have when deciding, for example, dialogue strategies, or keyword spotting (both of these will be discussed in other modules). For this practical the perfomance of the confidence scores is assessed using *Normalised Cross Entropy* (NCE), see Appendix E.

5 The HTK Toolkit

The HTK Toolkit consists of two main parts: a library of useful routines for accessing speech files, label files, HMM definitions etc; and a set of tools (i.e programs) for building and testing HMM recognisers [12]. In this practical, you will use the HTK tools to edit label files, adapt HMMs, rescore lattices and analyse the performance of your system. Descriptions of the tools referred to in this section are given in the HTK Book [1]. There is an PDF version of the HTK Book which can be accessed by typing

```
evince /usr/local/teach/MLSALT11/Speech/Docs/htkbook.pdf
```

(assuming you have got the appropriate display settings). Alternatively copy the file onto your laptop/workstation, and open using your PDF viewer. This section will focus on operations with HTK related to the practical.

5.1 Speaker Adaptation

In addition to training models, HTK supports various forms of linear model-based adaptation schemes. The linear transforms may either be estimated, or applied, using `HERest` or`HVite`. Additionally they can be applied using `HDecode`. To generate transforms the standard word-level transcriptions must be mapped to phone sequences. A script to first use `HVite` to generate the phone transcriptions, and then use `HERest` to estimate the transforms

```
/usr/local/teach/MLSALT11/Speech/scripts/hmmadapt.sh
```

An alternative approach is to use `HVite` when estimating the transforms. Transform estimation with `HVite` will be described in this section.

For both schemes, the first stage in generating linear transforms is to build a regression class tree, or specify the set of base-classes to use. A `global` base-class is supplied for th practical (and is the form used in the supplied script. To generate a regression class tree, suppose that an MMF containing all the models is in directory `hmm`.

```
HHEd -H hmm/MMF -M regtrees -w /dev/null edfile.rc2 models.lst
```

where the file `edfile.rc2` contains

```
LS "hmm/stats"
RC 2 "xform" {(sil,sp).state[1-10].mix}
```

will generate a regression class tree in file `regtrees/xform.tree` and associated base classes in `regtrees/xform.base`. This tree has two base classes, one consisting on the silence Gaussian component (`sil` and `sp`), the other all the speech Gaussian components. You should refer to the HTK manual to understand exactly what this command does. Using this regression class tree, running the command

```
HVite -a -H hmm/MMF -K xforms cmllr1 -J regtrees -C cmllr.cfg  \
    -h "%%%%%%%%%*" -I words.mlf -S train.scp words.dct models.lst
```

[12] General information about HTK can be found at the HTK website `http://htk.eng.cam.ac.uk`

will use the regression class tree generated, the word-level MLF and adaptation data to generate a CMLLR transform. The -K option specifies the directory in which the generated transform is to be stored. There is an optional second argument that specifies the file extension for the transform (in this case cmllr1). The config file cmllr.cfg contains the specification for the transform, for example

```
HADAPT:TRACE                    = 61
HADAPT:TRANSKIND                = CMLLR
HADAPT:USEBIAS                  = TRUE
HADAPT:REGTREE                  = xform.tree
HADAPT:ADAPTKIND                = TREE
HADAPT:SPLITTHRESH              = 1000.0
HADAPT:BLOCKSIZE                = "IntVec 3 13 13 13"
HMODEL:SAVEBINARY               = FALSE
HMODEL:TRACE                    = 512
```

This specifies a CMLLR transform with a block diagonal transform (specified using BLOCKSIZE)with a bias. The the number of transforms is determined using a regression class tree with a minimum occupancy of 1000.0 frames (specified using SPLITTHRESH. The -h option specifies that the first three letters of the filename are used to specify the files associated with the "speaker" and are used to determine where the file should be stored. If the file train.scp contained the following entries[13]

```
C001_show1_0001_1107.plp=/data/show1.plp[1,1107]
C001_show1_3130_4254.plp=/data/show1.plp[3130,4254]
C002_show1_1010_1507.plp=/data/show1.plp[1010,1507]
```

Two transform would be generated and stored in files

```
xforms/C001_show1.cmllr1
xforms/C002_show1.cmllr1
```

Note it is important the the scp file is ordered so that segments from the same cluster appear together. If show level transforms, rather than cluster level transforms, are required it is only necessary to change the -h option to ????_%%%%%*. Note the naming conventions for the files in this practical differ from the ones above. For a possible mask to use see the example adaptation script.

Transforms can be used as input and parent transforms. An input transform is used to improve the frame-state alignment (the transforms are estimated using EM). This acts in the same fashion as performing multiple training iterations when estimating acoustic models. To use the transform generated above as an input transform the command line is modified to

```
HVite -a -k -H hmm1/MMF -K xforms cmllr2 -J xforms cmllr1 -J regtrees \
    -C cmllr.cfg  -h "%%%%%%%%%*" -I words.mlf -S train.scp \
    words.dct models.lst
```

[13]Rather than breaking the data from a complete show into many separate files, HTK supports selecting sets of frames from one file in the scp file definition. This is the notation used in this example

This will produce files

```
xforms/C001_show1.cmllr2
xforms/C002_show1.cmllr2
```

This new transform will yield a higher (at least it will not decrease) the likelihood of the adaptation data compared to the original transform.

To make more "interesting" transforms, cascades of linear transforms (applied one after another) can also be trained. This is achieved using `parent` transforms. To generate an `MLLRMEAN` transform with the above `CMLLR` as the parent (and input transform) simply requires changing the command line to

```
HVite -a -k -H hmm1/MMF -K xforms mllr1 -J xforms cmllr2 -J regtrees \
    -E xforms cmllr2 -C mllr.cfg  -h "%%%%%%%%%%*" -I words.mlf \
    -S train.scp  words.dct models.lst
```

This will produce files

```
xforms/C001_show1.mllr1
xforms/C002_show1.mllr1
```

with transforms

```
xforms/C001_show1.cmllr2
xforms/C002_show1.cmllr2
```

as the parent transform - these are applied *before* the new transforms.

Having generated the transforms they can then be evaluated using, for example, `HVite` in a lattice rescoring mode[14]

```
HVite -k -H hmm1/MMF -i rescore.mlf -J xforms mllr1 -J regtrees \
      -w -L lattices -h "%%%%%%%%%%*" -I words.mlf \
      -C config.lat -S train.scp  words.dct models.lst
```

or with `HDecode` (as used in this practical)

```
HDecode -m -H hmm1/MMF -i rescore.mlf -J xforms mllr1 -J regtrees \
      -w -L lattices -h "%%%%%%%%%%*" -I words.mlf \
      -C config.lat -S train.scp  words.dct models.lst
```

This will use the cascade of a `CMLLR` and `MLLRMEAN` transform and will rescore the lattices in the directory `lattices`.

The code is designed to be highly flexible, allowing different combinations of transforms, regression trees, block structures etc.

5.2 Lattice Rescoring

In addition to performing decoding with language models `HTK` supports various operations on lattices. `HLRescore` is a tools for manipulating lattices. It reads lattices in standard lattice format (for example produced by `HVite`, see appendix A) and can apply one of the following operations:

[14] `HVite` can be used for decoding given a language model, however this mode will not be used in this practical

- find 1-best path through lattice: this allows language model scale factors and insertion penalties to be optimised rapidly;
- expand lattices with new language model: allows the application of more complex language, e,g, 4-grams, than can be efficiently used on the decoder;
- convert lattices to equivalent word networks: this is necessary prior to using lattices generated with `HVite` or `HDecode` to merge duplicate paths;
- calculate lattice statistics;
- prune lattice using forward-backward scores: efficient pruning of lattices to reduce development/evaluation rescoring times.

`HLRescore` expects lattices which are directed acyclic graphs (DAGs). If cycles occur in the lattices then `HLRescore` will throw an error. These cycles may occur after the merging operation (`-m` option) with HLRescore.

The following command will obtain the 1-best output from the lattices in `lattices` with a language model scale factor of 12.0 and insertion penalty of -10.0[15]

```
HLRescore -C config -f -s 12.0 -p -10.0 -i rescore.mlf \
-L lattices -S train.scp words.dct
```

To expand lattices with a new language in `tg.lm`[16] and generate the 1-best output using the same insertion penalty and grammar scale factor used to generate the lattice

```
HLRescore -C config -f -i rescore.mlf -L lattices -n tg.lm \
   -S train.scp words.dct
```

The following command will merge duplicate paths, (very aggressively) prune the lattice and output a new prune merged lattice in `lattices2`

```
HLRescore -w -q tvaldm -m f -t 50.0 100.0 -l lattices2 \
   -L lattices -S train.scp words.dct
```

In order to keep the disk-usage associated with storing the lattices to a minimum HTK supports the filters to compress, and decompress, lattices on the fly. The filters are specified in the configuration files. For this practical the configuration should contain the following lines

```
HNETFILTER   = 'gunzip -c $.gz'
HNETOFILTER  = 'gzip -c > $.gz'
```

[15] The language model scale factor and insertion penalty used to generate the lattices are contained in the lattice headers.

[16] See the HLM toolkit section for a discussion of uncompressing/compressing language models.

6 HLM Toolkit

The HTK Language Model (HLM) toolkit is distributed along the HTK toolkit. It allows a number of operations to be performed on language models and statistic obtained. The general use of the tools is the same as the HTK toolkit.

All the language models used in this practical are store in binary format and gzipped. In the same fashion as the lattices HTK/HLM supports the uncompressing and compressing language models on the fly. This is achieved using the following configuration variables

```
HLANGMODFILTER  = 'gunzip -cd $.gz'
HLANGMODOFILTER = 'gzip -c > $.gz'
```

Thus when accessing LMs the `.gz` suffix should be omitted from the filename.

The HLM tool `LPlex` can perform a number of operations that are useful for this practical. Using [17]

```
base/bin/LPlex -C lib/cfgs/hlm.cfg -u -t lms/lm1 lib/texts/dev03.dat
```

will compute the perplexity of the data in `lib/texts/dev03.dat` using the language model `lms/lm1`. When computing perplexities it is important to handle Out Of Vocabulary (OOV) words appropriately. Using the `-u` flag enables the tool to handle OOV words (which do occur in the practical and are important to quote when assessing language model performance). The format of the text data files is

```
<s> BUT DIDN'T ELABORATE </s>
```

It is important to include the sentence start and sentence end markers.

`LPlex` can also be run in a mode whereby streams of probabilities of word predictions are output to a file. This allows interpolation weights to be simply computed based on these streams of probabilities

```
base/bin/LPlex -C lib/cfgs/hlm.cfg -s stream -u -t lms/lm1 lib/texts/dev03.dat
```

will generate a stream of probabilities in file `stream` of the language model in `lms/lm1` predicting the word sequence in `lib/texts/dev03.dat`. Only non-OOV words are predicted. The configuration file is in `lib/cfgs/hlm.cfg`.

The second `HLM` tool that is used in this practical is `LMerge`. This allows a series of language models to be interpolated together given interpolation weights.

```
base/bin/LMerge -C lib/cfgs/hlm.cfg -i 0.6 lms/lm2 -i 0.1 lms/lm3 \
   lib/wlists/train.lst lms/lm1 lm_int
```

will combine `lms/lm1` with `lms/lm2` and `lms/lm3` with interpolation weights 0.3, 0.6 and 0.1 respectively and will store the merged language model in `lm_int.gz`. The word-list is in the file `lib/wlists/train.lst`.

For reference, as with HTK, HLM has a large number of other tools. See the HTK Manual for more details.

[17] These command lines can be directly run if links to the `lms`, `lib` and `base` directories in the main practical directory have been made.

M.J.F. Gales
October 2015

A HTK Standard Lattice Format

This is an example of the *standard lattice format* SLF that is used in HTK. The lattice is generated using a 20,000 word bigram language model. The best hypothesis is

```
<s> IT DIDN'T ELABORATE  </s>
```

the score for this path is -20218.25. Note the `<s>` and `</s>` model initial and final silence.

```
VERSION=1.1
UTTERANCE=4k0c030t
lmscale=16.0
wdpenalty=0.0
N=24   L=39
# Node definitions
I=0    t=0.00
I=1    t=0.25
I=2    t=0.26
I=3    t=0.61
I=4    t=0.62
I=5    t=0.62
I=6    t=0.71
I=7    t=0.72
I=8    t=0.72
I=9    t=0.72
I=10   t=0.72
I=11   t=0.72
I=12   t=0.72
I=13   t=0.73
I=14   t=0.78
I=15   t=0.78
I=16   t=0.80
I=17   t=0.80
I=18   t=0.81
I=19   t=0.81
I=20   t=1.33
I=21   t=2.09
I=22   t=2.09
I=23   t=2.85

# Link definitions.
#
J=0     S=0    E=1    W=<s>         v=0  a=-1432.27  l=0.00
J=1     S=0    E=2    W=<s>         v=0  a=-1500.93  l=0.00
J=2     S=0    E=3    W=<s>         v=0  a=-3759.32  l=0.00
J=3     S=0    E=4    W=<s>         v=0  a=-3829.60  l=0.00
J=4     S=1    E=5    W=TO          v=3  a=-2434.05  l=-87.29
J=5     S=2    E=5    W=TO          v=1  a=-2431.55  l=-87.29
J=6     S=4    E=6    W=AND         v=3  a=-798.30   l=-69.71
J=7     S=4    E=7    W=IT          v=0  a=-791.79   l=-62.05
J=8     S=4    E=8    W=AND         v=2  a=-836.88   l=-69.71
```

```
J=9     S=3    E=9    W=BUT         v=0  a=-965.47   l=-51.14
J=10    S=4    E=10   W=A.          v=0  a=-783.36   l=-105.95
J=11    S=4    E=11   W=IN          v=0  a=-835.98   l=-49.01
J=12    S=4    E=12   W=A           v=0  a=-783.36   l=-59.66
J=13    S=4    E=13   W=AT          v=0  a=-923.59   l=-77.95
J=14    S=4    E=14   W=THE         v=0  a=-1326.40  l=-27.96
J=15    S=4    E=15   W=E.          v=0  a=-1321.67  l=-121.96
J=16    S=4    E=16   W=A           v=2  a=-1451.38  l=-59.66
J=17    S=4    E=17   W=THE         v=2  a=-1490.78  l=-27.96
J=18    S=4    E=18   W=IT          v=0  a=-1450.07  l=-62.05
J=19    S=5    E=18   W=IT          v=0  a=-1450.07  l=-110.42
J=20    S=6    E=18   W=IT          v=0  a=-775.76   l=-85.12
J=21    S=7    E=18   W=IT          v=0  a=-687.68   l=-125.32
J=22    S=8    E=18   W=IT          v=0  a=-687.68   l=-85.12
J=23    S=9    E=18   W=IT          v=0  a=-687.68   l=-50.28
J=24    S=10   E=18   W=IT          v=0  a=-689.67   l=-108.91
J=25    S=11   E=18   W=IT          v=0  a=-706.89   l=-113.78
J=26    S=12   E=18   W=IT          v=0  a=-689.67   l=-194.91
J=27    S=13   E=18   W=IT          v=0  a=-619.20   l=-100.24
J=28    S=4    E=19   W=IT          v=1  a=-1567.49  l=-62.05
J=29    S=14   E=20   W=DIDN'T      v=0  a=-4452.87  l=-195.48
J=30    S=15   E=20   W=DIDN'T      v=0  a=-4452.87  l=-118.62
J=31    S=16   E=20   W=DIDN'T      v=0  a=-4303.97  l=-189.88
J=32    S=17   E=20   W=DIDN'T      v=0  a=-4303.97  l=-195.48
J=33    S=18   E=20   W=DIDN'T      v=0  a=-4222.70  l=-78.74
J=34    S=19   E=20   W=DIDN'T      v=0  a=-4235.65  l=-78.74
J=35    S=20   E=21   W=ELABORATE   v=2  a=-5847.54  l=-62.72
J=36    S=20   E=22   W=ELABORATE   v=0  a=-5859.59  l=-62.72
J=37    S=21   E=23   W=</s>        v=0  a=-4651.00  l=-13.83
J=38    S=22   E=23   W=</s>        v=0  a=-4651.00  l=-13.83
```

B HTK Confusion Network Format

An example of the format of the confusion network is shown below.

```
N=9
k=1
W=</s>                  s=2.45    e=3.51    p=0.00000
k=2
W=NEWS                  s=1.45    e=2.45    p=-0.00000
W=!NULL                 s=1.45    e=2.45    p=-29.80533
k=2
W=C.                    s=1.27    e=1.45    p=-0.00000
W=!NULL                 s=1.27    e=1.45    p=-18.54744
k=3
W=B.                    s=1.10    e=1.27    p=-0.01521
W=D.                    s=1.11    e=1.27    p=-4.51688
W=!NULL                 s=1.10    e=1.27    p=-5.47822
k=3
W=B.                    s=0.88    e=1.07    p=-0.45890
W=BE                    s=0.88    e=1.07    p=-1.00472
W=!NULL                 s=0.88    e=1.07    p=-6.28030
k=2
W=TO                    s=0.71    e=0.88    p=-0.00382
W=!NULL                 s=0.71    e=0.88    p=-5.57058
k=2
W=LISTENING             s=0.37    e=0.71    p=-0.00000
W=!NULL                 s=0.37    e=0.71    p=-13.56380
k=2
W=YOU'RE                s=0.21    e=0.37    p=-0.00180
W=!NULL                 s=0.21    e=0.37    p=-6.32362
k=1
W=<s>                   s=0.00    e=0.21    p=0.00000
```

Note these confusion networks are in reverse time order. Thus the 1-best output for this network is

```
<s> YOU'RE LISTENING TO B. B. C. NEWS </s>
```

`N` indicates the total number of score sets, `k` indicates the number of arcs in a set.

The timing information, `s` and `e` (start and end respectively), are given in seconds. The value associated with `p` is the log probability of that word.

C Language Model Information

LM	Source	Type	Size(MW)
lm1	PSM's BN transcripts 92-99 TDT2&TDT3 captions	newswire	275
lm2	Transcripts from CNN's website 99-00	newswire	66
lm3	TDT4 captions	acoustic	2
lm4	NIST's BN training data from 97/98 Marketplace show transcripts	acoustic	2
lm5	Newswire LAT and WP 95-98, NYT 97-00 Associated Press 97-00	newswire	674

Table 1: Language model training texts and their sizes.

Five tri-gram language models are provided for this practical. The source, type and size (in million words (MWs)) of the data used to train the language models is shown in table 1. For the type

- `acoustic`: transcriptions of spoken data
- `newswire`: mainly newspaper (and related) text

These sources are the same as those used in CU-HTK 2003 Broadcast News transcription system. For more details see [2]. For this practical the language models were more heavily pruned than those used in the full system, and the acoustic models are different to those described in the paper. Thus the baseline performance is not expected to be the same as the 19.7% WER shown in Table V in [2].

D Dictionaries

There are two forms of dictionaries used in this practical.

The first form of dictionary is based on phones. Here the actual pronunciation of the word is considered. Some example entries are:

```
A               ax
A               ey
A.              ey
A.'S            ey z
AAH             aa
```

You should be able to relate these pronunciations to the discussion of phones earlier in the course.

An alternative is to simply use the spelling of the word. Using the same words as above, the entries for the graphemic dictionary are:

```
A               a^I
A.              a^I;B
A.'S            a^I;BA s^F
AAH             a^I a^M h^F
```

in addition to the (lower-cased) spelling of the word, there are two additional sets of symbol. The first, after the caret, indicates the position in the word. The second set indicate any special attributes of the letter. For example `B` indicates that it is an abbreviation, and `A` it is followed by an apostrophe (or at the beginning of the word preceded by an apostrophe).

For some languages there is a clear mapping from the spelling to the pronunciation of the word. Unfortunately English is not one of these languages.

E Scoring Output

An example of the scoring output is given below

```
,-----------------------------------------------------------------------------------------.
|                  scoring/sclite/dev03sub_plp-bg_1best_LM12.0_IN-10.0.ctm                 |
|-----------------------------------------------------------------------------------------|
| SPKR                        | # Snt  # Wrd | Corr    Sub    Del    Ins    Err  S.Err |  NCE   |
|-----------------------------+--------------+---------------------------------------+--------|
| 20010117_2000_2100_pri_twd |   94    5022 | 87.4    9.5    3.0    2.7   15.3   92.6 | -0.856 |
|=========================================================================================|
| Sum/Avg                     |   94    5022 | 87.4    9.5    3.0    2.7   15.3   92.6 | -0.856 |
|=========================================================================================|
|           Mean              | 94.0  5022.0 | 87.4    9.5    3.0    2.7   15.3   92.6 | -0.856 |
|           S.D.              |  0.0    0.0  |  0.0    0.0    0.0    0.0    0.0    0.0 |  0.000 |
|          Median             | 94.0  5022.0 | 87.4    9.5    3.0    2.7   15.3   92.6 | -0.856 |
`-----------------------------------------------------------------------------------------'
```

The meaning of each of the columns are

- `SPKR`: name of the "speaker". In this case the data is labelled at the show level, so the "speaker" name is the show
- `# Snt`: number of sentences
- `# Wrd`: number of words
- `Corr`: percentage of correct words
- `Sub`: percentage of substitution errors
- `Del`: percentage of deletion errors
- `Ins`: percentage of insertion errors
- `Err`: percentage of word errors (100-%Accuracy)
- `S. Err`: percentage of sentence errors (100-%Accuracy)

The final column is the *normalized cross entropy* (NCE) score. This is a measure of how accurately the confidence scores reflect the actual probability of the word being correct.

The definition of the NCE score is

$$\texttt{NCE} = \frac{\mathcal{H}(\boldsymbol{c}) - \mathcal{H}(\boldsymbol{c}|\mathcal{M})}{\mathcal{H}(\boldsymbol{c})}$$

where

- $\mathcal{H}(\boldsymbol{c})$ is the entropy of the class labels (in this case `correct` and `incorrect`);
- $\mathcal{H}(\boldsymbol{c}|\mathcal{M})$ is the conditional entropy of the class labels using the confidence scores deived from the model $\mathcal{M}$.

The class labels are assumed to be binomial distributed and estimated using

$$\mathcal{H}(\boldsymbol{c}) = -\overline{p}\log(\overline{p}) - (1-\overline{p})\log(1-\overline{p})$$

An estimate needs to be made of the probability of the `correct` class label, $\overline{p}$. This is estimated from the recognition output (words $\boldsymbol{w}_{1:n}$).

$$\overline{p} \approx \frac{\texttt{number of tokens correctly recognized}}{\texttt{total number of tokens}}$$

Note confidence scores do not handle *deletions*.

The conditional entropy is approximated by its empirical estimate over the sequence $\boldsymbol{w}_{1:n} = \{w_1, \ldots, w_n\}$ with class labels $\boldsymbol{c}_{1:n} = \{c_1, \ldots, c_n\}$

$$\mathcal{H}(\boldsymbol{c}_{1:n}|\boldsymbol{w}_{1:n}, \mathcal{M}) \approx -\frac{1}{n}\left(\sum_{i=1}^{n} c_i \log(\hat{p}_i) + (1-c_i)\log(1-\hat{p}_i)\right)$$

where

- $\hat{p}_i$ is the confidence score that word i is correct (estimated from the model $\mathcal{M}$)
- c_i is the label whether word i is correct ($c_i = 1$) or incorrect ($c_i = 0$)

The NCE score has a maximum value of 1 and a minimum value of $-\infty$