Automatic Speech Recognition 2008-09: Lab-session sheet (5) — Continuous speech recognition (part 3 of 3) —

Hiroshi Shimodaira (*Revision* : 1.1)

In this lab-session you will

- try triphone models: word-internal triphones, state-clustered word-internal triphones
- train N-gram language models and evaluate them by means of perplexity and WER

1 Word-internal triphone models

1.1 Updating your environment

To make your experimental environment up-to-date, please re-run the initialisation command we used last week:

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

1.2 Preparation

We at first need to

- convert the current monophone dictionary (`resources/dictionaries/mono.dct`) into a triphone one (`./resources/dictionaries/tri.dct`)
- create a list of word-internal context-dependent triphone models
- create triphone label files for training.

For this, run the following script

```
./scripts/mk_triphone_labels
```

which will create

```
./resources/dictionaries/tri.dct
./model_lists/tri.list
./labels/phone/tri.mlf
```

Take a look at those files to understand what information they contain.

1.3 Single-mixture word-internal triphone models (R5)

We then create initial triphone models by simply cloning monophones. Run the following script for this

```
./scripts/mk_triphone_models
```

which will run HHEd with the editing commands

```
MM "trP_" { *.transP }
CL "model_lists/tri.list"
```

to create initial triphone models in `models/R5/hmm0`.

Run the following script to re-train the models,

```
./scripts/train_triphones 0 1 2 3
```

which will create `./models/R5/hmm{1,2,3,4}`.

Now you are ready to run a recognition experiment with the following script

```
./scripts/recognise_with_triphone_models 4
```

This will use the most recent models, `./models/R5/hmm4`, and recognition results are saved in `./recognised/R5/`.

As is mentioned in the previous handouts, if the output of the script, i.e. HVite, contains the message "`No tokens survived to final node of network`", meaning HVite failed to recognise the data, you need to change the parameters in the script to reduce the message, otherwise recognition accuracy shown by the script `./scripts/results_summary` might be underestimated.

In order to check the occurrence of the errors above, it is a good idea that you redirect the standard output of the script into a file (say, "`log`"), and run the following script to see how many those errors are reported in the log file

```
./scripts/check-no-tokens log
```

1.4 Tied-mixture word-internal triphone models (R6) ··· optional

Although the recognition performance of the triphone models trained in the last section could be improved by increasing the number of mixtures of each state as was done for monophone models in the previous lab session, there would be a certain danger that the training will suffer from the data sparseness problem, i.e. shortage of training samples. For this reason, we will employ tied-mixture approach in this section, followed by next section in which we will try state-shared approach.

To create a initial set of triphone models, run the following script

```
./scripts/mk_tm_triphones
```

which will call HHEd with the following editing commands stored in `./resources/config/clone-mmix.hed`

```
MM "trP_" { *.transP }
SS 4
JO 128 2.0
TI MIX_1_ {*.state[2-4].stream[1].mix}
JO 128 2.0
TI MIX_2_ {*.state[2-4].stream[2].mix}
JO 128 2.0
TI MIX_3_ {*.state[2-4].stream[3].mix}
JO 32 2.0
TI MIX_4_ {*.state[2-4].stream[4].mix}
HK TIEDHS
CL "model_lists/tri.list"
```

Train the model with the script:

```
./scripts/train_triphones -R6 0 1 2
```

1.5 State-clustered word-internal triphone models (R8)

Here, we will use models in `R5/hmm4` as the initial set of models to carry out the bottom-up state clustering.

1.5.1 Single-mixture models

Run the following script for this clustering:

```
./scripts/mk_sc_triphones
```

which will produce single-mixture state-clustered triphones under `models/R8`.
Next, run the following script to retrain the models.

```
./scripts/train_sc_triphones 0 1 2 3
```

For recognition, use the following script

```
./scripts/recognise_with_sc_triphone_models 4
```

You will need to change the parameter(s) in the file to reduce the "no token survived" messages.

1.5.2 Multiple-mixture models

The mixing-up operation is done in the same way as was done for monophone models in the previous lab, but we will use the following another script, which will mix-up the single-mixture models in `R8/hmm4` to 2-mixture ones and save them into `R8/hmm10`.

```
./scripts/mixups_sc_triphones -i 4 -o 10 -m 2
```

Retraining should be run as

```
./scripts/train_sc_triphones 10 11 12 13
```

Evaluate the `R8/hmm14` in the same manner with the one for the single-mixture models.
Repeat the procedure above with varying the mixture number gradually. Note that you need to appropriately change the arguments of the scripts above in the repetition.

2 Language Models

2.1 Preparation

```
% cd ~/asr/ASR
% mkdir LM
% cd LM
% ~hshimoda/pub-asr/bin/init-t5
```

2.2 Building and evaluating language models

The CMU-Cambridge Statistical Language Modelling Toolkit v2 is used to train and evaluate n-gram models.

Basically, users have to run three commands with several arguments in the toolkit to get a n-gram model. For simplicity, this session provides scripts to build and test a bigram. Read the scripts to understand what they do, make sure your are at the right current working directory, i.e. `~/asr/ASR/LM`, then run them:

```
% ./scripts/make_lm
% ./scripts/test_lm
```

The `make_lm` command reads the corpus `data/training.text` and generates a language model `models/`N`gram.arpa` in ARPA format, where N is the order of the language model.

Make sure you understand exactly what perplexity is and how it is computed. Modify **both** scripts so you can build and test models of other orders (N=2,3,...6). Note that you will get an error if you try and build a 1-gram (to fix this, edit the options to `idngram2lm` in `scripts/make_lm` – remove the `-context` flag and its argument).

- Plot perplexity against model order. Look at the actual language model files. What do you notice?
- How does language model order affect speed (of both building and testing). What implications does that have for speech recognition?

- Now **read the manual** for the tools available as

  `~/asr/ASR/Org/manuals/CMU_LM_TOOLKIT/toolkit_docuentation.html`

  and try varying other options, like the backing-off strategy and the thresholds. Can you make the language models better (and what does "better" mean) ?
- Again, make plots of perplexity against LM order. Keep notes, particularly about the best **bigram** model you manage to build – you will need to refer back to them later.

2.3 Converting the language models to HTK format

HTK provides a tool, `HBuild`, for converting ARPA-format **unigram** or **bigram** models into the lattice (finite-state) format required by `HVite`. A script, "scripts/arpa2htk", is available for you to do this conversion. Its functions are as follows:

1. Call `HBuild`.

```
% HBuild -u "<unk>" -s "<s>" "</s>" \
  -n models/2gram.arpa data/wordlist models/2gram.lat
```

2. Edit the resulting `2gram.lat` file and change `<s>` to `!SENT_START` and `</s>` to `!SENT_END`.

Before you run the script, make sure you know what parameters you used to build the language models there (if not, just re-build them) and do:

```
% ./scripts/arpa2htk
```

You will find a language model, `models/wp.net`. Take a look at it to see what conversion has been made.

Now, you need to copy the language model to the directory which is refereed by your recogniser. You can do this by this way:

```
% cp -p models/wp.net ../resources/language_models
```

Next, return to the original directory for ASR experiments by

```
% cd ..
```

and edit your recognition script, `resources/language_models/`, to replace the current language model file, `net.wp`, with the new one, `wp.net`.

Run the decoder again (you may want to also play with the language model scaling factor) and compare the accuracy to your previous results.