Lab 8: Training triphone models

University of Edinburgh

March 23, 2020

Path errors

If you get errors such as `command not found`, try sourcing the path again:

```
source path.sh
```

In general, every time you open a new terminal window and `cd` to the work directory, you would need to source the path file in order to use any of the Kaldi binaries.

There is an appendix at the end of every lab with the most typical mistakes.

0.1 Triphone models

In Lab 2, you trained monophone models, but there's lots of room for improvement. Let's start training a triphone model with delta and delta-delta features.

Backslashes

Backslashes in Bash (\) which is present in the next box, are simply a way of splitting commands over multiple lines. That is, the following two commands are identical:

```
some_script.sh --some-option somefile.txt
```

and

```
some_script.sh --some-option \
  somefile.txt
```

You may remove the backslash and type these commands on a single line. But sometimes when writing scripts, avoiding too long commands on a single line can help readability.

Be careful about spaces after \. Bash will expect a newline immediately, and a space here before the newline will make a script crash.

Run the following commands to train a triphone system. This might take a while...

```
steps/align_si.sh data/train_2kshort \
  data/lang exp/mono exp/mono_ali

steps/train_deltas.sh --nj 4 2500 15000 data/train_2kshort \
  data/lang exp/mono_ali exp/tri1
```

- While that is running, open another terminal window, change directory to the work directory and source the path.

0.2 Delta features

Above we started running a script called `train_deltas.sh`. This trains triphone models on top of `MFCC+delta+deltadelta` features. To avoid having to store features with `delta+deltadelta` applied, Kaldi adds this in an online fashion, just as another pipe, using the programme `add-deltas`. Remember how we checked the feature dimension of the features in the first lab? Run the following command.

```
feat-to-dim scp:data/train/feats.scp -
```

What do you expect the dimension to be be after applying `add-deltas`? Run the following command.

```
add-deltas scp:data/test/feats.scp ark:- | feat-to-dim ark:- -
```

0.3 Logs

Kaldi creates detailed logs during training. These can be very helpful when things go wrong. By now we should have some of the first ones created for the triphone training:

```
less exp/tri1/log/acc.1.1.log
```

Notice that on the top, the entire command which Kaldi ran (as set out by the script) is displayed. For this example it runs a command called `gmm-acc-stats-ali`, and then if you look closely there is a feature pipeline using the programmes `apply-cmvn` and `add-deltas` which applies these transforms and additions to the features in an online fashion.

0.4 Triphones

When we ran the triphone modelling script above we also passed two numbers, 2500 and 15000. These are respectively the number of leaves in the decision tree and the total number of Gaussians across all states in our model.

Triphone clustering

As you may recall from the lectures, having a separate model for each triphone is generally not feasible. With typically 48 phones we would require $48 \times 48 \times 48$, i.e. more than 110000 models. We don't have enough data to see all those, so we cluster them using a decision tree. The *number of leaves* parameter then sets the maximum number of leaves in the decision tree, and the *number of gaussians* the maximum number of Gaussians distributed across the leaves. So on average our model will have an average of $\frac{numgauss}{numleaves}$ Gaussians per leaf.

To see how many states have been seen, run the following command:

```
sum-tree-stats --binary=False - \
  exp/tri1/1.treeacc | head -1
```

The first number indicates the number of states with statistics. Dividing that number by three will roughly give the number of seen triphones (why is this?).

Let's have a closer look at the clustering in Kaldi. It's also a good opportunity to look a bit deeper at the Kaldi scripts. Open the training script we just ran by typing

```
less steps/train_deltas.sh
```

At the top is a configuration section with several default parameters. These are the parameters that the script can take by passing `--param setting` to the script when running it. Most scripts in Kaldi are set up this way. The first one is a variable called `stage`, this can be really useful to start a script partway through. Scroll down till line 88 which says

```
if [ $stage -le 2 ]; then
```

This is saying that if the stage variable is less than or equal to two, run this section. This is the section that builds the decision tree. It first calls a binary called `cluster-phones`, this uses e.g. k-means clustering to cluster similar phones. These clusters will be the basis for the questions in the decision tree. Kaldi doesn't use predefined questions such as "is the left phone a fricative?", but rather estimates them from the data. It writes these using the numeric phone identities to a file called `exp/tri1/questions.int`. Let's look at it using a utility that maps the integer phone identities to their more readable names. Leave `less` by pressing `q` and run the next command:

```
utils/int2sym.pl data/lang/phones.txt \
  exp/tri1/questions.int | less
```

Each line is a cluster. Some of these clusters are really large, but hopefully some of them should make sense. We use these to build a clustering tree, stored in `exp/tri1/tree`. Exit `less` and run the following command:

```
copy-tree --binary=false exp/tri1/tree - | less
```

This file sets out the entire clustering tree. You can read more about how the decision tree is encoded at http://kaldi-asr.org/doc/tree_externals.html#tree_example

Once the triphone system has finished training, we can decode our test data again, this may take a few minutes:

```
utils/mkgraph.sh data/lang_test_bg \
  exp/tri1 exp/tri1/graph
```

```
steps/decode.sh --nj 4 exp/tri1/graph \
  data/test exp/tri1/decode_test
```

While waiting, have a look again at some of the densities:

```
gmm-copy --binary=false exp/tri1/final.mdl - |\
  python2.7 local/plot_gmm.py -
```

How have they changed from the monophone system? What would happen if we modelled this data using single Gaussians instead?

When the decoding have finished, score the directory by running:

```
local/score.sh data/test exp/tri1/graph \
  exp/tri1/decode_test
```

We can then have a look at the WER by typing:

```
more exp/tri1/decode_test/scoring_kaldi/best_wer
```

That should be considerably better than the monophone system, but there's still lots of room for improvement, even before moving to neural network based systems. The typical progression in Kaldi would now be to train a system on top of decorrelated features and then train a system with speaker adaptive training. But that's beyond the scope of this lab.

0.5 Appendix: Common errors

- Forgot to source `path.sh`, check current path with `echo $PATH`
- No space left on disk: check `df -h`
- No memory left: check `top` or `htop`
- Lost permissions reading or writing from/to AFS: run `kinit && aklog`. To avoid this, run long jobs with the `longjob` command.
- Syntax error: check syntax of a Bash script without running it using `bash -n scriptname`
- Avoid spaces after \when splitting Bash commands over multiple lines
- Optional params:
- command line utilities: `--param=value`
- shell scripts: `--param value`
- Most file paths are absolute: make sure to update the paths if moving data directories
- Search the forums: http://kaldi-asr.org/forums.html
- Search the old forums: https://sourceforge.net/p/kaldi/discussion

0.6 Appendix: UNIX

- `cd dir` - change directory to `dir`, or the enclosing directory by `..`
- `cd -` - change to previous directory
- `ls -l` - see directory contents

- `less script.sh` - view the contents of `script.sh`
- `head -l` and `tail -l` - show first or last `l` lines of a file
- `grep text file` - search for `text` in `file`
- `wc -l file` - compute number of lines in `file`