Lab 4: Hybrid Acoustic Models

University of Edinburgh

February 12, 2019

In this lab we will look at training hybrid neural network acoustic models using a frame-level cross-entropy loss. We will continue using the word level models from the last lab for alignments. The lab will be done in a Google Cloud instance with access to GPUs.

Path errors

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

```
source path.sh
```

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

1 Setup

Let's begin by opening a terminal window and setting up a Google Cloud instance. We have packaged most of the commands for you. Follow the instructions on screen.

```
cd ~/asrworkdir
source path.sh
gcloud auth login
```

Next, create an instance of an image we have prepared by running the following command. Replace sxxxxxx with your UUN.

```
local/create_gcloud.sh sxxxxxx-asrlab
```

GPU quota

If the above command fails, please try the instructions in the answer provided here to increase the GPU quota for your account:
`https://stackoverflow.com/a/49737435`

Finally, ssh to your new image:

```
gcloud compute ssh --zone "us-west1-b" "asr-lab4"
```

Shutting down the instance

Important: Before you leave the lab, make sure to stop the instance with the command below. Otherwise the funds associated with your account may run out.

```
gcloud compute instances stop asr-lab4
```

To start it again later:

```
gcloud compute instances start asr-lab4
```

Once in the image, navigate to asrworkdir and source the path as usual.

```
cd /home/asrworkdir
source path.sh
```

1.1 Alignments

Before we train the neural networks, lets try to get better alignments by making a stronger acoustic model.

Align the system from the last lab by running

```
steps/align_si.sh --nj 4 data/train_words \
  data/lang_wsj exp/word/tri1 exp/word/tri1_ali
```

Alignments

Why do we have a separate step to align between model training? In Kaldi, we typically do a pass of alignment between each training phase. This is just to ensure that we have the absolute latest alignments for the latest model in each stage. Technically we could skip these parts, and most scripts that take alignment experiment directories will also accept a normal training experiment directory (e.g. `tri1` vs. `tri1_ali`).

Train a system on top of LDA+MLLT features, using the `tri1_ali` alignments:

```
steps/train_lda_mllt.sh \
  --splice-opts "--left-context=3 --right-context=3" \
  2500 15000 data/train_words data/lang_wsj \
  exp/word/tri1_ali exp/word/tri2
```

LDA+MLLT

LDA and MLLT transformations have not been covered in the course. We provide a very brief introduction here. For more information see [1][a].

We first splice together 7 frames (`left` and `right-context=3` above) of the MFCC features. The dimensionality is then reduced to 40 using Linear Discriminant Analysis (LDA) using the acoustic states as classes. How much of a dimensionality reduction is this, given the 13-dimensional MFCCs and the context window?

Finally, during training we estimate a transform known as Maximum Likelihood Linear Transform (MLLT). Don't confuse this with Maximum Likelihood Linear Regression (MLLR), which we use for adaptation. Normally in HMM-GMM systems, we model the emission probabilities using diagonal covariance matrices for the GMMs. This is because there would otherwise be far too many parameters to estimate. However, modelling without covariances assumes that each element of the feature vectors (e.g. MFCCs) are independent. MLLT is a way to loosen this assumption somewhat, by sharing a few full covariance matrices across many distributions. This models some of the covariances, without the corresponding explosion in the number of parameters.

[a] `http://mi.eng.cam.ac.uk/~sjy/papers/gayo07.pdf`

Finally, align the system once more so that we have the latest possible alignments for the neural networks:

```
steps/align_si.sh --nj 4 data/train_words \
  data/lang_wsj exp/word/tri2 exp/word/tri2_ali
```

2 Neural networks

Kaldi comes with three neural network toolkits. We will use `nnet1`[1]. First, let's separate the data into training and validation data. The script `utils/subset_data_dir_tr_cv.sh` by default separates data into 90% for training and 10% for validation:

[1] `http://kaldi-asr.org/doc/dnn1.html`

```
dir=data/train_words
utils/subset_data_dir_tr_cv.sh $dir ${dir}_tr90 ${dir}_cv10
```

This will have created these directories:

```
data/train_words_tr90
data/train_words_cv10
```

We will begin training a fairly small neural network model. Feel free to experiment with the number of hidden layers and the hidden dimension. These settings are certainly not optimal!

```
steps/nnet/train.sh --hid-layers 6 --hid-dim 512 --splice 5 \
  --learn-rate 0.008 \
  data/train_words_tr90 data/train_words_cv10 data/lang_wsj \
  exp/word/tri2_ali exp/word/tri2_ali exp/word/nnet
```

Runtime

If we were to run this on a CPU, this will likely take longer than the lab time to complete. Training on a CPU is at least 10-20x slower.

While this is running, open another terminal window, ssh to the server and look at the training script (if you're familiar with screen or tmux you can also just make another window in the existing session):

```
cd ~/asrworkdir && source path.sh
gcloud compute ssh --zone "us-west1-b" "asr-lab4"
cd /home/asrworkdir
less steps/nnet/train.sh
```

Scroll down to the section that begins with "`###### PREPARE ALIGNMENTS`". There are two important events occurring in this section, we first generate target labels using the alignments we created above. Then, we generate counts of the PDFs corresponding to the phones in the alignments. Convince yourself of where this is happening in the code.

Let's have a closer look at both cases.

2.1 Labels

Look at the labels by running the following command

```
ali-to-pdf exp/word/tri2_ali/final.mdl \
  "ark:gunzip -c exp/word/tri2_ali/ali.1.gz |" ark:- | \
   ali-to-post ark:- ark,t:- | less
```

Can you relate the first number in each bracket to a previous lab? This file sets out, for each frame, the phone state which will be on. Or equivalently, which output of the neural network will be set to 1[2]. This makes sense since we will only expect one phone state to be active at any given time. The second number is a weight, which for our cases will always be set to 1.0. Leave `less` by hitting `q`.

2.2 PDF counts

Next, run the following command.

```
ali-to-phones --per-frame=true exp/word/tri2_ali/final.mdl \
  ark:"gunzip -c exp/word/tri2_ali/ali.1.gz |" ark:- | \
  analyze-counts --verbose=1 ark:- -
```

Recall from the lectures the theory on hybrid acoustic models. We still make use of Hidden Markov Models (HMMs), however, we now replace the GMMs used to estimate output pdfs by the outputs of neural networks. That is, we want to train the neural network to classify phones, and given the output probabilities, we want to compute the likelihood of the state q given the features $\mathbf{x}$, $p(\mathbf{x}|q)$. We can interpret the output of the neural network as the probability of a phone class given the feature $p(q|\mathbf{x})$. Then, using Bayes rule we get *scaled* likelihoods:

$$\frac{p(q|\mathbf{x})}{p(q)} = \frac{p(\mathbf{x}|q)}{p(\mathbf{x})}, \tag{1}$$

where on the left hand side we have divided by the class priors. We don't get $p(\mathbf{x}|q)$ exactly, but this is fine since $p(\mathbf{x})$ does not depend on the class q. For more information, see lecture 8.

All this means that we will have to scale the outputs by the class priors, in order to use the neural network outputs in place of the GMMs. The `train.sh` script computes these from the alignments and stores them in a file called `ali_train_pdf.counts`, using a command similar to the one above. Let's see how this plays out when decoding. Open the corresponding decoding script by running

```
less steps/nnet/decode.sh
```

Search for "counts" by typing a forward slash and then "counts":

```
/counts
```

and hit enter.

We see first that we can provide the counts if we'd like. Hit `n` a couple of times. There are now a couple of lines which look for particular files. Note that

[2] The rest of the outputs are set to zero. Remember that we are doing frame cross-entropy training.

we fall back to `ali_train_pdf.counts` if we can't find anything else. This is the file created above. Hit `n` a few times more and you will get to a line that begins with `nnet-forward`. This is a forward pass that will be used to provide the class probabilities for decoding. Notice that the class frame counts are passed in as an argument.

Let's try to look at the output of a forward pass. First, as in the scripts, lets set up a feature stream:

```
# first set up the original features
feats="ark:copy-feats scp:data/train_words/feats.scp ark:- |"

# then splice with 5 context frames on either side
feats="$feats splice-feats --left-context=5 \
  --right-context=5 ark:- ark:- |"
```

Then run a forward pass and look at the output. What happens if you remove the prior counts?

```
nnet-forward \
 --class-frame-counts=exp/word/tri3_nnet/ali_train_pdf.counts \
 exp/word/tri3_nnet/final.nnet "$feats" ark,t:- | less
```

3 Architecture

There's one last, important piece of the puzzle. When we ran `steps/nnet/train.sh` above we specified some parameters for the network, such as the number of hidden layers, the layer width, etc. What this does is actually to create a prototype file, setting out the overall architecture. Look at it by typing

```
less exp/word/tri3_nnet/nnet.proto
```

Can you work out what each part means? Why is the input dimension 143? Recall that we ran the training script with `--splice 5`. For more interesting architectures, it can sometimes be worth modifying this file directly. It can then be passed to the training script using `--nnet-proto my.proto`. Let's go ahead and modify the prototype. First, copy the file:

```
cp exp/word/tri3_nnet/nnet.proto my.proto
```

Now open the file in a text editor, such as `nano`, `vim`, `emacs` or `gedit`, e.g.:

```
nano my.proto
```

Try changing some parameters (for example change input layer to take 13 units and the first layer to have 5 neurons). Finally, let's generate an initial model with all the required parameters:

```
nnet-initialize --binary=false my.proto my.init
```

If that was successful you can look at the model by typing

```
nnet-info my.init
```

Then, go ahead go ahead and open the file itself to see it in full detail:

```
less my.init
```

Can you relate the parameters you set to what you observe in the model file?

Finally let's decode the model. First we need to create a graph on which to decode (more on this in a later lecture):

```
utils/mkgraph.sh data/lang_wsj_test_bg exp/word/tri2 \
exp/word/tri2/graph
```

Then decode and score:

```
steps/nnet/decode.sh --nj 4 exp/word/tri2/graph \
data/test_words exp/word/nnet/decode_test

local/score_words.sh data/test_words \
exp/word/tri1/graph exp/word/nnet/decode_test
```

Check the WER by running

```
more exp/word/nnet/decode_test/scoring_kaldi/best_wer
```

Finally, exit the ssh session (ctrl+D) and make sure to stop the instance to save the rest of the funds in your account.

```
gcloud compute instances stop asr-lab4
```

3.1 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`

3.2 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`

References

[1] Mark Gales and Steve Young. The application of hidden markov models in speech recognition. *Foundations and trends in signal processing*, 1(3):195–304, 2008.