Speech Recognition

- a practical guide

Lecture 2

Hidden Markov Models

First, a detour...

Nazi Germany, 1939

The Enigma machine

The Enigma

- Enigma was one of many such machines
- Encrypted text messages sent by radio
- Substituted letters in an order-dependent way (a disk rotated with each letter, changing the pattern of substitutions)
- Had a large number of settings controlled by rotors
- Recipient had to know the rotor settings

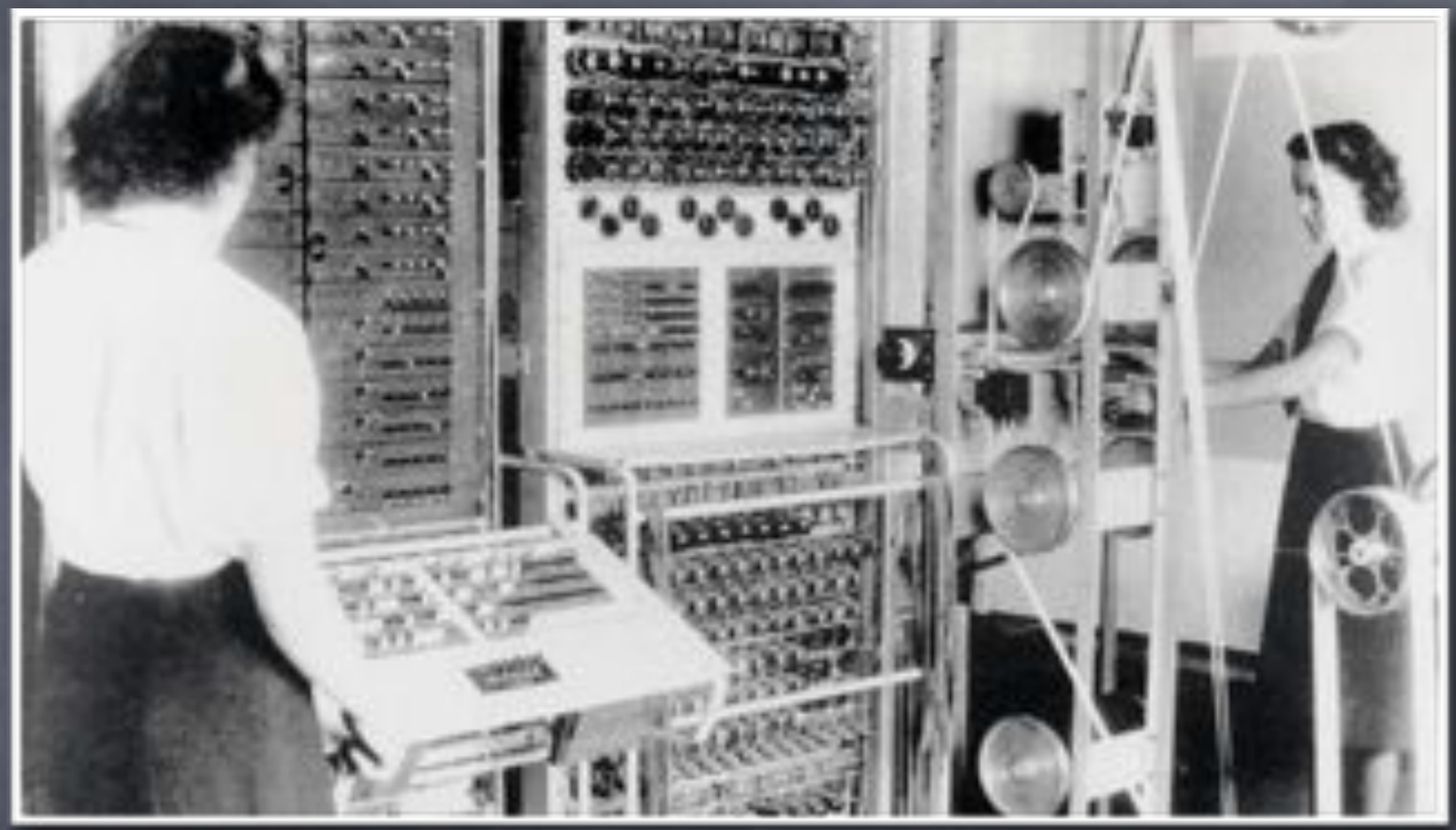
Secret!

Geheim!
Nicht ins Flugzeug mitnehmen!

Sonder-Maschinenschlüssel BGT

Datum	Walzenlage	Ringstellung	Steckerverbindungen	Kenngruppe
31.	I V III	06 20 24	UA PF RQ SO NI EY BG HL TX ZJ	jeu nyq aqm
30.	V II III	01 07 12	GF KV JM IB UW LX TD QS NA ZH	azs zds kck
29.	IV I V	11 17 26	CI OK PV ZL HX NB AW DJ FE ST	kap gwh lyx

- Rotor settings were changed daily (the same everywhere).
- The Allies could use computers to try out each combination of rotor settings.
- For the correct combination, the decrypted messages would "make sense"



Colossus

the first programmable computer
(Bletchley Park, England)

Models of text

- A computer was used to try out the various combinations fast.*
- But wait!
- A computer cannot tell whether message makes sense
- Need to have a model of (German) text
- This model will assign a higher score to sequences that look like "real text"

*Disclaimer: this is all a gross oversimplification of what really happened.

Simplest model

- Assume letters produced "independently" (i.i.d. = identically and independently distributer).
- Just use letter frequencies, e.g. "e" more common.
- For each letter, P(letter) is relative frequency.
- P(sentence) is product over letters of P(letter)

Drawbacks

- That model is sufficient to break simple codes (e.g. a rotation of the letters).
- Enigma machine was not a simple substitution or rotation cipher.
- Many different rotor settings would produce plausible letter frequencies
  - e.g. swapping 2 letters of similar frequency

Markov Chain

- A Markov Chain is a model of probabilities of sequences of symbols
- In 1st order Markov chain, p(symbol) depends only on previous symbol.
- E.g. p(hat) = p(h|?) p(a|h) p(t|a)
- Write p(h|?) because we're not addressing start/end effects right now.
- Would work out a table of probabilities from previous telegrams.

Hidden Markov Model (HMM)

- A model of the probability of a sequence.
- At each time instant, model is in some "hidden" state.
- Matrix of "emission probabilities": #states by #symbols
- Matrix of "transition probabilities": #states by #states
- Note: invented by US govt. researchers, probably for some code-breaking stuff (but the exact use has not been made public.)

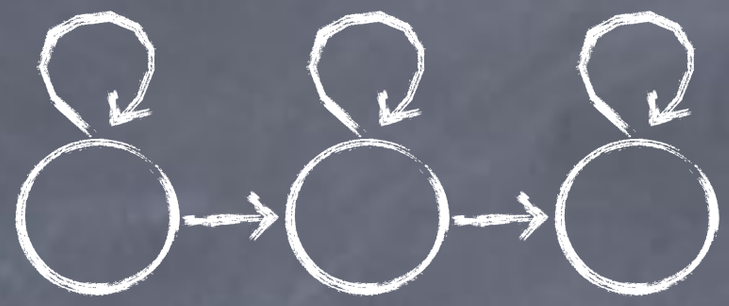
HMMs

- Computing p(sequence | model) involves summing over exponentially many state sequences
- Can be done fast using a dynamic programming recursion.
- Training the model parameters aims to maximize train-data likelihood.
  - Not just a question of computing frequencies like for Markov chain.
- You have to work out the hidden state sequences (well, a distribution over them)

HMMs: important algorithms

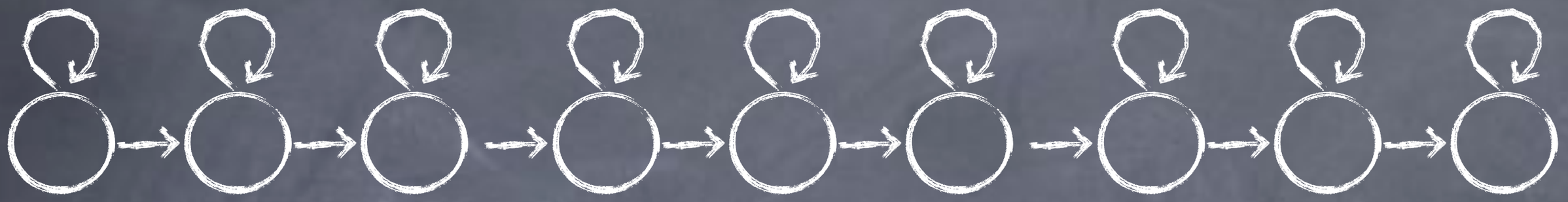
- Forward-backward algorithm
  - Recursion to compute state occupation probs.
  - Used during model training.
    - This is a class of algorithms for iteratively maximizing likelihood.
- Viterbi algorithm
  - Finds most likely sequence of HMM states given a symbol sequence.

HMMs for speech



- Old HMM-based speech-reco used to work like this:
  - Use Vector Quantization (VQ) to map each speech feature to one symbol (out of typically around 256).
  - Each phone has a 3-state HMM with a left-to-right structure as above.

HMMs for speech (2)



- The model for a sentence is a concatenation of the models for its phones.

- You don't need phone time alignment to train.
- The forward-backward algorithm just finds the right alignment itself, after many iterations.
  - This relies on there being enough training sentences (and nice enough data)

Continuous HMMs

- Vector Quantization was done by training a Gaussian Mixture Model (GMM) and using the top-scoring index.

gaussian mixture model

- After some time people switched to a "soft" Vector Quantization: sum over the index, rather than take the max.
- Eventually the Gaussians were made specific to the HMM-states.
- This is a "continuous" HMM, not discrete: the states emit a continuous feature vector, not a discrete symbol.

Things you should know

- The following are very fundamental things that you should learn if you don't know already:
  - Markov Chain
  - Hidden Markov Model
  - Forward-backward algorithm
  - Viterbi algorithm
  - E-M for mixture of Gaussians

Monophone model training

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ ## these commands are in run.sh
$ . path.sh  ## set up your path-- will be needed later.
$ scripts/subset_data_dir.sh data/train 1000 data/train.1k
$ steps/train_mono.sh data/train.1k data/lang exp/mono
$ local/decode.sh --mono steps/decode_deltas.sh exp/mono/decode
```

- Assumes you are where you left off last week
- See kaldi.sf.net for slides
- Note: "monophone" is to distinguish from phonetic-context-dependent HMMs "triphones"

Monophone model training

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ ## these commands are in run.sh
$ scripts/subset_data_dir.sh data/train 1000 data/train.1k
$ steps/train_mono.sh data/train.1k data/lang exp/mono
$ local/decode.sh --mono steps/decode_deltas.sh exp/mono/decode
```

- Use a subset of data since waste of time to use all of it (so few parameters to train)
- Suggested exercise: try with different amounts of data and see how WER changes
  - Does WER change smoothly?

Monophone model training

```
$ cd ~/kaldi-trunk/egs/rm/s3
$ ## these commands are in run.sh
$ scripts/subset_data_dir.sh data/train 1000 data/train.1k
$ steps/train_mono.sh data/train.1k data/lang exp/mono
$ local/decode.sh --mono steps/decode_deltas.sh exp/mono/decode
```

- Next we'll look at how the training script works.
- Output is in exp/mono
- Will look at log files to show you the commands the script actually runs.

Cepstral normalization

```
$ cat exp/mono/cmvn.log
compute-cmvn-stats --spk2utt=ark:data/train.1k/spk2utt scp:data/train.1k\
/feats.scp ark:exp/mono/cmvn.ark
```

- For each speaker, compute statistics to normalize the means and variances of the cepstral features.
- Just count, and (sum, sum-squared) for each dim.
- Statistics are in binary format

Viewing CMVN stats

```
$ copy-matrix ark:exp/mono/cmvn.ark ark,t:- | head
copy-matrix ark:exp/mono/cmvn.ark ark,t:-
adg0  [
  202805.7 -45171.13 -25113.25 -32178.11 -64940.17 -59834.14 -53859.89
-25581.98 -38369.97 -35770.74 -34123.73 -40811.7 -17615.15 3120
  1.370877e+07 1926569 693237.5 1436362 2282034 1924576 1983340 791023.9
1075897 947114.2 995890.7 1152457 501246.2 0 ]
ahh0  [
  243559.7 -49834.79 -45551.43 -32294.71 -31345.15 -69359.74 -21432.35
-62158.46 -2514.336 -19708.29 -40365.65 37920.28 -36189.33 3720
  1.676417e+07 2100356 1245457 1481856 963323.5 2106689 816228.5 2149422
596726.6 830598.1 1141586 1077821 1064480 0 ]
```

- Part highlighted in yellow went to stderr
- All logging goes to stderr (including echoing command line).
- This is standard archive of matrices.

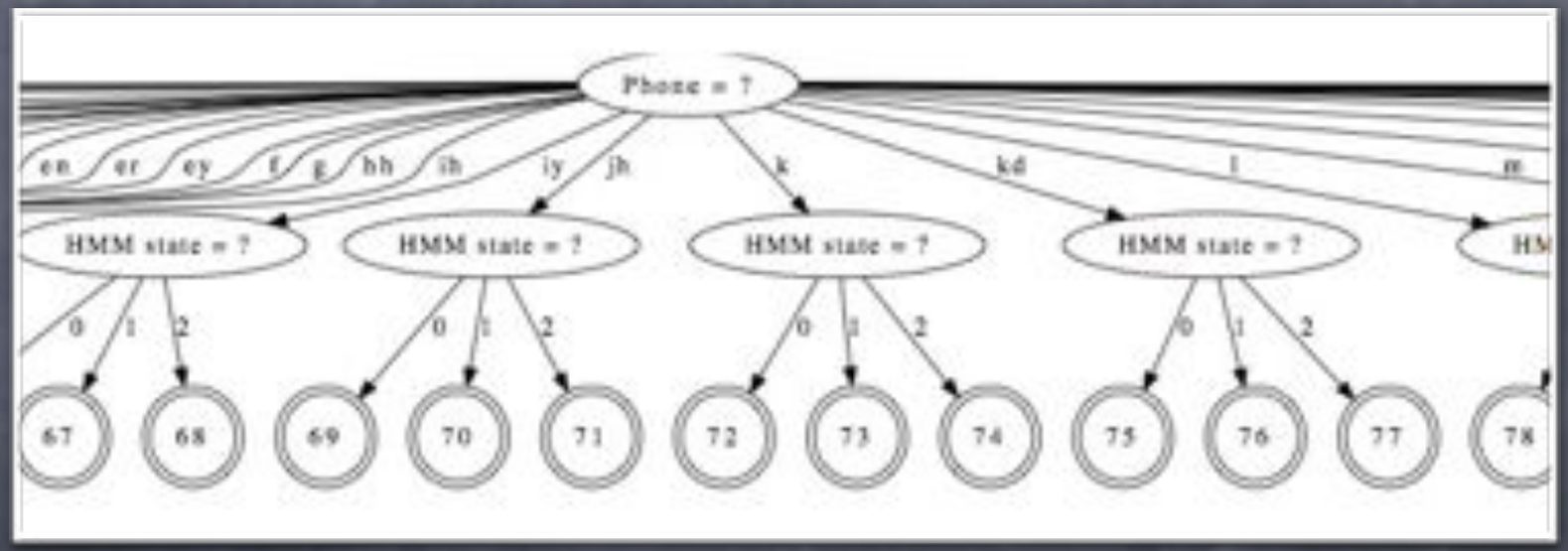
Model initialization

```
$  head exp/mono/init.log
gmm-init-mono '--train-feats=ark:apply-cmvn --norm-vars=false --
utt2spk=ark:data/train.1k/utt2spk ark:exp/mono/cmvn.ark scp:data/train.1k/
feats.scp ark:- | add-deltas ark:- ark:- | subset-feats --n=10 ark:-
ark:-|' data/lang/topo 39 exp/mono/0.mdl exp/mono/tree
```

- (ignore part in gray; used to get plausible means and variances)
- Input: topology file data/lang/topo, and dim (39)
- Outputs: model "0.mdl" and "tree"
  - Tree is phonetic-context decision tree-- doesn't have any splits in monophone case.

Monophone tree

```
$  draw-tree data/lang/phones.txt exp/mono/tree | \
   dot -Tps -Gsize=8,10.5 | ps2pdf - ~/tree.pdf
```



- whole set of trees represented as one tree
- ask first about central phone.

Viewing topology file

```
$ cat data/lang/topo
<Topology>
<TopologyEntry>
<ForPhones>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46 47
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
</TopologyEntry>
<TopologyEntry>
<ForPhones>
48
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.25 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3
0.25 </State>
```

- Specifies 3-state left-to-right HMM, and default transition probs (before training)
- Separate topology for silence (5 states, more transitions)

Compiling training graphs

```
$  cat  exp/mono/compile_graphs.log
compile-train-graphs exp/mono/tree exp/mono/0.mdl data/lang/L.fst ark:exp/
mono/train.tra 'ark:|gzip -c >exp/mono/graphs.fsts.gz'
LOG (compile-train-graphs:main():compile-train-graphs.cc:150) compile-
train-graphs: succeeded for 1000 graphs, failed for 0
```

- Compiles FSTs, one for each train utterance
- Encode HMM structure for that training utterance
- We precompile them because otherwise this would dominate training time

Viewing training graphs

```
$ fstcopy 'ark:gunzip -c exp/mono/graphs.fsts.gz|' ark,t:- | head
fstcopy 'ark:gunzip -c exp/mono/graphs.fsts.gz|' ark,t:-
trn_adg04_sr249
0       1       266     949     0.693359
0       106     284     0       0.693359
0       107     285     0       0.693359
0       108     286     0       0.693359
1       7       268     0
1       1       265     0
2       109     288     0
2       110     289     0
2       111     290     0
```

- Archive format is: (utt-id graph utt-id graph...)
- Graph format is:
  - from-state to-state input-symbol output-symbol cost
  - Costs include pronunciation probs, but for training graphs, not transition probs (added later).

Symbols in graphs

- In graphs for training and testing...
  - Output symbols are words (look up in words.txt)
  - In the traditional recipe, input symbols would be p.d.f.'s (so each mixture of Gaussians has a number)
- This causes difficulties for training the transition probabilities and finding phone alignments etc.
- In our graphs, input-symbols are "transition-ids", which correspond roughly to arcs in context-dependent HMMs. See docs!
  - Can be mapped to "pdf-ids" which are fewer.

First alignment stage

```
$ Extract from steps/train_mono.sh
align-equal-compiled "ark:gunzip -c $dir/graphs.fsts.gz|" "$feats" \
   ark:-  2>$dir/align.0.log | \
 gmm-acc-stats-ali --binary=true $dir/0.mdl "$feats" ark:- \
     $dir/0.acc 2> $dir/acc.0.log
```

- Produces alignments "equally spaced" for each utterance, accumulates 1st iteration stats.
- An alignment is a vector of ints (per utterance)
- Note: we do Viterbi training not forward-backward
  - means we use 1-best path.

Looking at alignments

```
$ head -1 exp/mono/cur.ali
trn_adg04_sr249 285 283 283 283 283 283 283 283 283 291 292 292 292 292 292 292
290 300 299 266 265 265 265 268 267 270 269 269 269 14 16 18 230 229 232 231 234
146 145 145 145 148 147 150 149 149 149 149 149 104 103 106 108 107 107 107 194
196 195 195 198 197 32 31 34 33 36 35 44 43 43 43 46 45 48 128 130 129 129 132
158 157 157 157 157 157 157 157 157 160 162 161 161 32 34 33 36 74 76 78 77 77
77 77 77 134 133 133 133 133 133 133 133 133 133 133 136 135 135 138 278 277 277
277 277 277 280 279 279 279 282 281 56 58 57 57 57 60 32 31 34 36 278 280 282
218 220 219 222 221 221 221 221 221 221 221 221 221 221 221 221 221 98 97 97 97 97
97 97 97 100 99 102 164 166 165 165 168 167 32 34 36 170 172 174 173 173 173 122
121 121 124 123 123 123 123 126 125 8 7 10 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 12
260 259 262 264 263 263 263 263 263 263 263 284 283 283 283 290 287 287 287 287
287 287 300 299
```

- These are the alignments we got after training the model more.

Looking at alignments

```
$ show-alignments data/lang/phones.txt exp/mono/30.mdl  ark:exp/mono/cur.ali | head -2
show-alignments data/lang/phones.txt exp/mono/30.mdl ark:exp/mono/cur.ali
trn_adg04_sr249  [ 285 283 283 283 283 283 283 283 283 291 292 292 292 292 292 292 290 300 299 ]
[ 266 265 265 265 268 267 270 269 269 269 ] [ 14 16 18 ] [ 230 229 232 231 234 ] [ 146 145 145 145
148 147 150 149 149 149 149 149 ] [ 104 103 106 108 107 107 107 ] [ 194 196 195 195 198 197 ] [ 32
31 34 33 36 35 ] [ 44 43 43 43 46 45 48 ] [ 128 130 129 129 132 ] [ 158 157 157 157 157 157 157 157
157 160 162 161 161 ] [ 32 34 33 36 ] [ 74 76 78 77 77 77 77 77 ] [ 134 133 133 133 133 133 133 133
133 133 133 136 135 135 138 ] [ 278 277 277 277 277 277 280 279 279 279 282 281 ] [ 56 58 57 57 57
60 ] [ 32 31 34 36 ] [ 278 280 282 ] [ 218 220 219 222 221 221 221 221 221 221 221 221 221 221 221
221 ] [ 98 97 97 97 97 97 97 97 100 99 102 ] [ 164 166 165 165 168 167 ] [ 32 34 36 ] [ 170 172 174
173 173 173 ] [ 122 121 121 124 123 123 123 123 126 125 ] [ 8 7 10 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
12 ] [ 260 259 262 264 263 263 263 263 263 263 263 ] [ 284 283 283 283 290 287 287 287 287 287 287
300 299 ]
trn_adg04_sr249  sil                                                                               w
ah            td                      k                                                        ey
p                            ax                     b                         ih                    l
ax               dx                            iy
z                                                  d                     ax               z
sh                                                                  er
```

- The program "show-alignments" displays them in more readable format, with phones.

First update

```
$ cat exp/mono/update.0.log
gmm-est --min-gaussian-occupancy=3 --mix-up=250 exp/mono/0.mdl exp/mono/
0.acc exp/mono/1.mdl
LOG (gmm-est:Update():transition-model.cc:374) TransitionModel::Update,
objf change is 0.109912 per frame over 348500 frames; 0 probabilities
floored, 0 states skipped due to insuffient data.
LOG (gmm-est:main():gmm-est.cc:101) Transition model update: average
0.109912 log-like improvement per frame over 348500 frames.
LOG (gmm-est:main():gmm-est.cc:109) GMM update: average 0.507126 objective
function improvement per frame over 348500 frames.
LOG (gmm-est:SplitByCount():am-diag-gmm.cc:179) Split 146 states with
target = 250, power = 0.2, perturb_factor = 0.01 and min_count = 20, split
#Gauss from 146 to 250
LOG (gmm-est:main():gmm-est.cc:140) Written model to exp/mono/1.mdl
```

- Note: Gaussians allocated proportional to small power of state occupation count (0.2 by default)

Looking at models

```
$ gmm-copy --binary=false exp/mono/30.mdl - | head
gmm-copy --binary=false exp/mono/30.mdl -
<TransitionModel>
<Topology>
<TopologyEntry>
<ForPhones>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.75 <Transition> 1 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.75 <Transition> 2 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 2 0.75 <Transition> 3 0.25 </State>
<State> 3 </State>
```

- Model file contains transition-model object, then GMM object
- Transition-model object is model-type-independent

Model building schedule

```
$ less steps/train_mono.sh  # snipped it a bit and expanded some variables...
x=1
while [ $x -lt 30 ]; do
  echo "Pass $x"
  if echo 1 2 3 4 5 6 7 8 9 10 12 15 20 25 | grep -w $x >/dev/null; then
    echo "Aligning data"
    gmm-align-compiled $scale_opts --beam=$beam --retry-beam=$[$beam*4] \
      $dir/$x.mdl "ark:gunzip -c $dir/graphs.fsts.gz|" "$feats" \
        ark,t:$dir/cur.ali 2> $dir/align.$x.log || exit 1;
  fi
  gmm-acc-stats-ali --binary=false $dir/$x.mdl "$feats" ark:$dir/cur.ali $dir/
$x.acc 2> $dir/acc.$x.log  || exit 1;
  gmm-est --write-occs=$dir/$[$x+1].occs --mix-up=$numgauss $dir/$x.mdl $dir/
$x.acc $dir/$[$x+1].mdl 2> $dir/update.$x.log || exit 1;
  rm $dir/$x.mdl $dir/$x.acc $dir/$x.occs 2>/dev/null
  if [ $x -le $maxiterinc ]; then
     numgauss=$[$numgauss+$incgauss];  # We'll get to 1000 Gaussians eventually
  fi
  x=$[$x+1]
done
```

Model building schedule

```
$ less steps/train_mono.sh  # snipped it a bit and expanded some variables...
x=1
while [ $x -lt 30 ]; do
  echo "Pass $x"
  if echo 1 2 3 4 5 6 7 8 9 10 12 15 20 25 | grep -w $x >/dev/null; then
    echo "Aligning data"
    gmm-align-compiled $scale_opts --beam=$beam --retry-beam=$[$beam*4] \
      $dir/$x.mdl "ark:gunzip -c $dir/graphs.fsts.gz|" "$feats" \
        ark,t:$dir/cur.ali 2> $dir/align.$x.log || exit 1;
  fi
  gmm-acc-stats-ali --binary=false $dir/$x.mdl "$feats" ark:$dir/cur.ali $dir/
$x.acc 2> $dir/acc.$x.log  || exit 1;
  gmm-est --write-occs=$dir/$[$x+1].occs --mix-up=$numgauss $dir/$x.mdl $dir/
$x.acc $dir/$[$x+1].mdl 2> $dir/update.$x.log || exit 1;
  rm $dir/$x.mdl $dir/$x.acc $dir/$x.occs 2>/dev/null
  if [ $x -le 20 ]; then
     numgauss=$[$numgauss+37];  # We'll get to 1000 Gaussians eventually
  fi
  x=$[$x+1]
done
```

Features

```
$ less steps/train_mono.sh  # snipped it a bit and expanded some variables...
<snip>
feats="ark:apply-cmvn --norm-vars=false --utt2spk=ark:$data/utt2spk ark:$dir/
cmvn.ark scp:$data/feats.scp ark:- | add-deltas ark:- ark:- |"

<snip>

  gmm-acc-stats-ali --binary=false $dir/$x.mdl "$feats" ark:$dir/cur.ali $dir/
$x.acc 2> $dir/acc.$x.log
```

- For monophone stage, use "delta" and "acceleration" features (add-deltas); dim now 39
  - Deltas computed over 5-frame window, like HTK
- Shell variable $feats specifies features.

Testing

```
$ local/decode.sh --mono steps/decode_deltas.sh exp/mono/decode
exp/mono/decode/wer_10:%WER 11.896593 [ 1491 / 12533, 45 ins, 606 del, 840 sub ]
exp/mono/decode/wer_4:%WER 9.726323 [ 1219 / 12533, 133 ins, 289 del, 797 sub ]
exp/mono/decode/wer_5:%WER 9.853985 [ 1235 / 12533, 109 ins, 331 del, 795 sub ]
exp/mono/decode/wer_6:%WER 10.213038 [ 1280 / 12533, 84 ins, 393 del, 803 sub ]
exp/mono/decode/wer_7:%WER 10.460385 [ 1311 / 12533, 74 ins, 431 del, 806 sub ]
exp/mono/decode/wer_8:%WER 10.819437 [ 1356 / 12533, 68 ins, 480 del, 808 sub ]
exp/mono/decode/wer_9:%WER 11.362004 [ 1424 / 12533, 58 ins, 544 del, 822 sub ]

$ ## You can get this same output by doing:
$ grep WER exp/mono/decode/wer_*
```

- Decoding script generates lattices
- These are rescored with different acoustic scales and all the WERs are printed out.
- We generally quote the best one
  - It's considered more proper to use a "dev set".

Decoding script

```
$ # the script
$ # local/decode.sh --mono steps/decode_deltas.sh exp/mono/decode
$ # calls:
$ scripts/mkgraph.sh --mono data/lang_test exp/mono exp/mono/graph
$ # then for each test set, e.g.:
$ steps/decode_deltas.sh exp/mono data/test_feb89 data/lang exp/mono/decode/feb89
```

- Note: "decoding" refers to the computation where we find the best sentence given the model.

Decoding output

```
$ less exp/mono/decode/feb89/decode.log
gmm-latgen-simple --beam=20.0 --acoustic-scale=0.1 --word-symbol-table=data/lang/words.txt exp/
mono/final.mdl exp/mono/graph/HCLG.fst 'ark:compute-cmvn-stats --spk2utt=ark:data/test_feb89/
spk2utt scp:data/test_feb89/feats.scp ark:- | apply-cmvn --norm-vars=false --utt2spk=ark:data/
test_feb89/utt2spk ark:- scp:data/test_feb89/feats.scp ark:- | add-deltas ark:- ark:- |' 'ark:|
gzip -c > exp/mono/decode/feb89/lat.gz' ark,t:exp/mono/decode/feb89/test.tra ark,t:exp/mono/
decode/feb89/test.ali
add-deltas ark:- ark:-
compute-cmvn-stats --spk2utt=ark:data/test_feb89/spk2utt scp:data/test_feb89/feats.scp ark:-
apply-cmvn --norm-vars=false --utt2spk=ark:data/test_feb89/utt2spk ark:- scp:data/test_feb89/
feats.scp ark:-
feb89_cmh18_st0058 WHAT+S DENVER ARE REASONER+S RATING AND DESTINATION OF SACRAMENTO
LOG (gmm-latgen-simple:main():gmm-latgen-simple.cc:218) Log-like per frame for utterance
feb89_cmh18_st0058 is -10.2242 over 366 frames.
feb89_cmh18_st0128 WHOSE AVERAGE SPEED OF AJAX GREATER THAN EIGHTEEN KNOTS
LOG (gmm-latgen-simple:main():gmm-latgen-simple.cc:218) Log-like per frame for utterance
feb89_cmh18_st0128 is -10.1073 over 322 frames.
feb89_cmh18_st0190 HOW SOON WILL FLINT GET PARAMETER ON ASW MISSION READINESS
LOG (gmm-latgen-simple:main():gmm-latgen-simple.cc:218) Log-like per frame for utterance
feb89_cmh18_st0190 is -10.2132 over 364 frames.
```

- Note: the sub-processes also print their own command-line arguments (explains first few lines)
- This output is just for debug (also gives lattice)

Homework

- This is optional...
- Run the steps described in this lecture
- Do one of:
  - Test whether varying the subset size makes a difference
  - See if using PLP features helps (see run.sh)
  - Try different options to the "add-deltas" program (note: train and test must match)
- Email your results to dpovey1@jhu.edu

End of this lecture