ASR with Kaldi Tutorial

Gilles Boulianne[1] Vishwa Gupta[1] Jan Trmal[2] Jérôme Labonté[1] Simon Desrochers[1]

Presented at École de Technologie Supérieure, Montréal, June 10, 2019

[1]Centre de recherche Informatique de Montréal

[2]Johns Hopkins University

Overview

Schedule

09:00 AM to 10h20 AM	Fundamentals
10:20 AM to 10h40 AM	*Break*
10:40 AM to 11h30 AM	Case-study: Air Traffic Control Challenge
11:30 AM to 12h00 PM	Prepare and launch your own ASR
12:00 PM to 01:30 PM	*Lunch Break*
01:30 PM to 03:00 PM	ASR System Building Lab
03:00 PM to 03:30 PM	*Coffee break*
03:30 PM to 05:00 PM	Debugging and error analysis

Part I

Fundamentals

Outline

Introducing Kaldi

Automatic Speech Recognition

Weighted Finite-State Transducers

Next steps

Introducing Kaldi

Introducing Kaldi

Kaldi is a toolkit for voice-related applications

- Speech recognition
- Speaker recognition
- Speaker diarisation

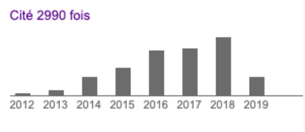
Important features

- C++ library, command-line tools, scripts.
- BLAS and LAPACK routines, CUDA GPU implementation.
- Licensed under Apache 2.0, not restrictive.
- Recipes for building speech recognition systems with widely available databases.
- Pre-trained models publicly released.

Kaldi today

Kaldi began in a JHU workshop in Baltimore, 2009.

- Community of Researchers Cooperatively Advancing ASR
- Top ASR performance in open benchmark tests
  - NIST OpenKWS ('14), IARPA ASpIRE ('15), MGB-3 ('17)
- Widely adopted in academia and industry
  - 2900+ citations up to now based on Google scholar data
  - Used by several US and non-US companies
- Main "trunk" maintained by Johns Hopkins
  - Forks contain specializations by JHU and others



From: Jan Trmal et al., "Kaldi ASR Tutorial for SLTU 2018"

Kaldi's team

Co-PI's, PhD Students and Sponsors

- Sanjeev Khudanpur
- Daniel Povey
- **Jan Trmal**
- L. Paola Garcia
- Mahsa Yarmohammadi
- Pegah Ghahremani
- Vimal Manohar
- David Snyder
- Yiming Wang
- Hainan Xu
- Xiaohui Zhang
- and **several others**

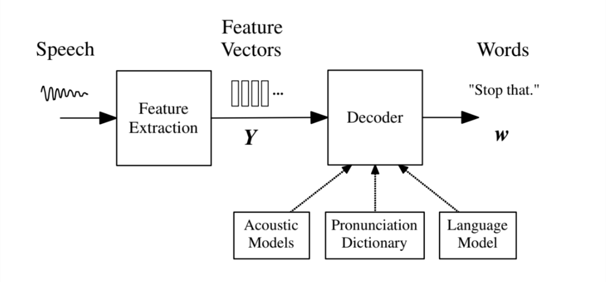
Office of the Director of National Intelligence
I A R P A
BE THE FUTURE

From: Jan Trmal et al., "Kaldi ASR Tutorial for SLTU 2018"

Automatic Speech Recognition

ASR: From Sound to Computation

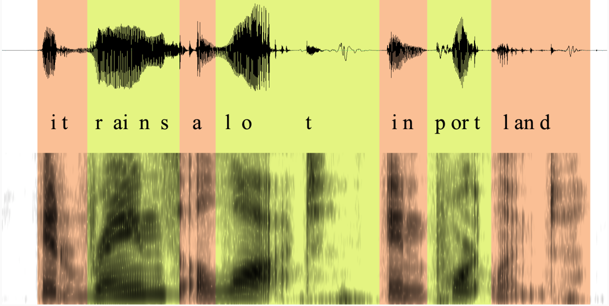
Sound in, computation, words out.



From: Gales and Young, The Application of Hidden Markov Models", 2007.

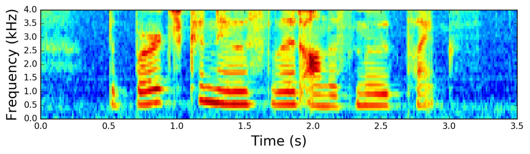
ASR: Time-frequency representation

Spectrogram: perceptual experiments, speech synthesis show that it represents content, speaker identity, emotion, etc.

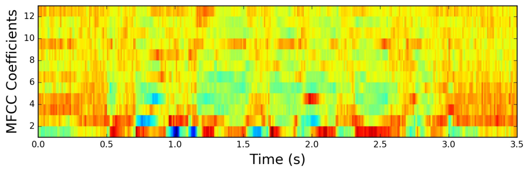


ASR: From sound to feature vectors

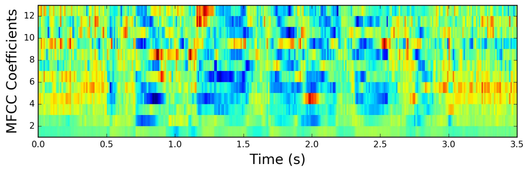
Filter banks



MFCCs

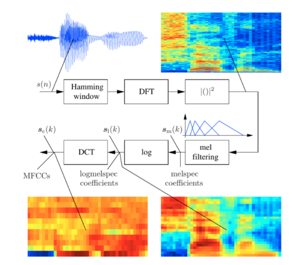


CMVN



ASR: Feature vectors

From spectrogram and images, feature vector representation.



From: A. Sehr, "Reverberant Modeling for Robust Distant-Talking Speech Recognition," 2010

Matrix [time x D]:

$$\boldsymbol{O} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T] = \boldsymbol{o}_t \qquad \text{for} \quad t = 1, \ldots, T$$

where $\boldsymbol{o}_t$ is a feature vector in $\mathcal{R}^{\mathcal{D}}$, one every 10 ms.

ASR: Maximum Likelihood

Maximum likelihood approach: not only one, but most successful.

Given:

- $\boldsymbol{O}$: a sequence of "observations" (feature vectors)
- $P(W \mid \boldsymbol{O})$: the probability distribution of a word sequence given an observation sequence

$\boldsymbol{O} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T]$

$W = [w_1, w_2, \ldots, w_N]$

Find $\hat{W} = argmax_W P(W \mid \boldsymbol{O})$

Estimate $P(W \mid \boldsymbol{O})$ directly: pattern recognition approach.

Not successful in the past, now revisited as end-to-end modeling.

ASR: ASR equation

Baye's rule:

$$P(W \mid \boldsymbol{O}) = \frac{P(\boldsymbol{O} \mid W)P(W)}{P(\boldsymbol{O})} \tag{1}$$

$$\hat{W} = argmax_W P(\boldsymbol{O} \mid W)P(W) \tag{2}$$

for a given $\boldsymbol{O}$.

Called a generative model: $W \rightarrow \boldsymbol{O}$.

Sub-problems:

- Estimate model
- Find best W that maximizes $\underbrace{\boxed{P(\boldsymbol{O} \mid W)}}_{\text{acoustics}}$ $\underbrace{\boxed{P(W)}}_{\text{language}}$.

ASR: Language Model

$P(W)$ in ASR equation.

Chain rule allows the decomposition:

$$P(w_i, \dots, w_1) = P(w_i \mid w_{i-1}, \dots, w_1) \cdot P(\overbrace{w_{i-1}, \dots, w_1}^{\text{state}}) \tag{3}$$

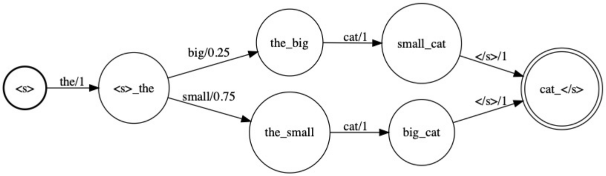
$$P(s_i) = P(w_i \mid s_{i-1}) P(s_{i-1}) \tag{4}$$

N-grams: limit history to previous N words (Markov property):

$$P(w_{i-1}, \dots, w_1) \quad \approx \quad P(w_{i-1}, \dots, w_{i-N})$$

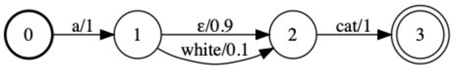
Useful because: compact representation, regular grammar.

ASR: Graph representation



$$P(\text{the}, \text{big}, \text{cat}) = 1.0 \times 0.25 \times 1.0 = 0.25 \quad (5)$$

$$P(\text{the}, \text{small}, \text{cat}) = 1.0 \times 0.75 \times 1.0 = 0.75 \quad (6)$$



$$P(\text{a}, \text{white}, \text{cat}) = 1.0 \times 0.10 \times 1.0 = 0.10 \quad (7)$$

$$P(\text{a}, \text{cat}) = 1.0 \times 0.90 \times 1.0 = 0.90 \quad (8)$$

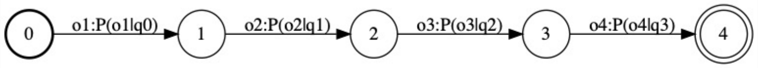
ASR: Acoustic Model

$P(\boldsymbol{O} \mid W)$: chain rule, and independence assumption.

Boils down to:

$$P(\boldsymbol{O} \mid Q)P(Q) = \prod_{i=1}^{T} P(\boldsymbol{o}_i \mid q_i) \times \prod_{i=1}^{T} P(q_i \mid q_{i-1})$$

where $Q = [q_1, q_2, \ldots, q_T]$ is a sequence of "acoustic states" (one per frame).



How do we compute $P(\boldsymbol{o}_t \mid q_i)$?

ASR: Gaussian Mixture Model

Gaussian distribution pdf:

$$P(\boldsymbol{o}_t) = \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$$

Simplistic, cannot account for variability in speaker, channel, etc.

Mixture of Gaussian pdf:

$$P(\boldsymbol{o}_t) = \sum_{c=1}^{C} \phi_c \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_c, \boldsymbol{\sigma}_c^2)$$

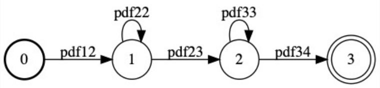
Parameters for each pdf: $\phi_c, \boldsymbol{\mu}_c, \boldsymbol{\sigma}_c$ for $c = 1, \ldots, C$

Final form: $P(\boldsymbol{o}_t \mid pdf_i)$

ASR: Hidden Markov Model

Still missing: $P(Q \mid W)$ which maps from words to acoustic states (one per frame).

Spoken words have varying length.



pdf12 pdf23 pdf34

pdf12 pdf22 pdf23 pdf34

pdf12 pdf22 pdf22 ... pdf23 pdf34

pdf12 pdf23 pdf33 pdf34

pdf12 pdf23 pdf33 ... pdf34

etc.

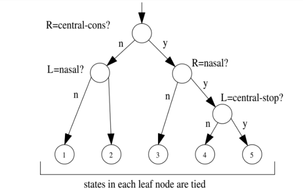
ASR: Decision Tree

Monophones: context-independent phones.

Triphones: phones with one left-context phone and one right-context phone.

```
W AY T -> W+AY W-AY+T AY-T
```

Decision tree: clusters several triphones sharing one pdf.



From: Gales and Young, The Application of Hidden Markov Models", 2007.

ASR: Wrap up

We now have a way of computing prob. of an observation sequence given any word sequence.

$P(\boldsymbol{O} \mid W) = \sum_{Q} P(\boldsymbol{O} \mid Q)P(Q \mid W)P(W)$

And the best path:

$$\hat{W} = argmax_{w} P(\boldsymbol{O} \mid W)P(W) \tag{9}$$

is the solution that maximizes the prob (our original problem):

$$P(W \mid \boldsymbol{O}) = \frac{P(\boldsymbol{O} \mid W)P(W)}{P(\boldsymbol{O})} \tag{10}$$

ASR: Finding most probable sequence

Graph representation: probability of a sequence = product of probabilities on a path.

Most probable sequence is given by most probable path in graph.

Graph search algorithms:

- depth-first: A-star, ...
- breadth-first: Viterbi beam search, token passing, ...

ASR: problem solved?

How to get probabilities in language and acoustic models?

Estimate from data: training the models.

For good results, training require lots of data, and lots of computation.

So far, acoustic model based on words. Need to train each word.

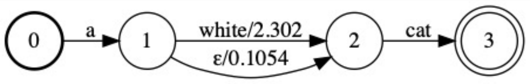
Each new task, each change in vocabulary requires retraining the model.

Lots of variation for a given word, difficult to capture.

Need to complicate our models a little bit, but we'll need more mathematical tools.

Weighted Finite-State Transducers

WFST: Weighted Finite State Automata



Finite state automata with labels and weights.

Example: language model.

In Kaldi, most common weight type is minus log probability.

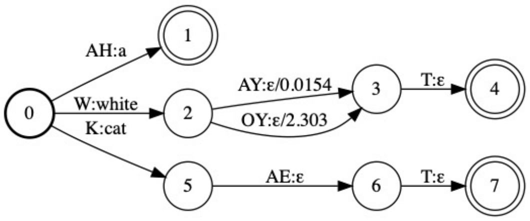
Cost (length) of a path: sum of arc weights.

Union of paths: min of arc weights.

Best probability path = shortest path.

Main operators: intersection, minimization.

WFST: Weighted Finite State Transducers



Finite state automata with input labels, output labels and weights.

Maps sequences of input labels to sequences of output labels.

Example: pronunciation lexicon.

Maps phoneme sequences to word sequences, with pronunciation probability.

WFST: Composition $\circ$

An operation that combines 2 WFSTs.

F_a with input A and output B.

F_b with input B and output C.

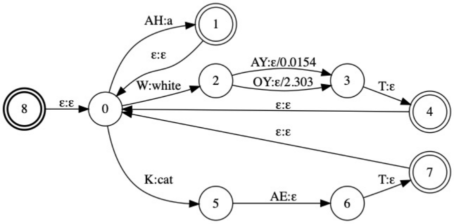
$F_a \circ F_b$ maps input A to output C.

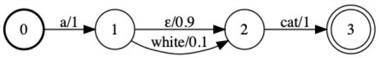
Generalization of intersection (think in sets of sequences).

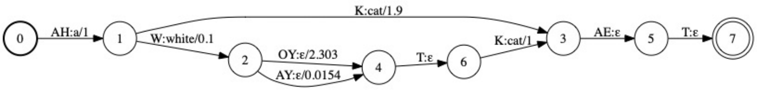
Weights are combined according to probability rules (in -log domain).

WFST: Composition example

Example: phonemes to word sequences.







WFST: Problem decomposition

So far, we have a word-based ASR model $w \rightarrow \boldsymbol{o}$

Phoneme-based model? Only 30 to 60 units instead of thousands.

Phonemes are easier to model and more flexible:

$$w \rightarrow p \rightarrow \boldsymbol{o}$$

Coarticulation modeled with phonemes-in-context:

$$w \rightarrow p \rightarrow pic \rightarrow \boldsymbol{o}$$

We will model phonemes-in-context with GMM pdf's:

$$w \rightarrow p \rightarrow pic \rightarrow pdf \rightarrow \boldsymbol{o}$$

WFST: Standard decomposition

- G (grammar): maps words to word sequence with LM probs.
- L (lexicon): maps phonemes sequences to words, with pronunciation probs.
- C (context): maps phonemes-in-context to phoneme sequences.
- H (HMM): maps phonemes-in-context to pdf id sequences.
- O (observations): provides $P(\boldsymbol{o}_t \mid pdf_i)$

$$\hat{W} = \mathrm{bestpath}(O \circ H \circ C \circ L \circ G)$$

WFST: Comments on the approach

Why WFSTs instead of just coding each model.

- Take advantage of FST theory and powerful mathematical tools
- Most concepts in ASR can be understood in terms of WFST
- Existing libraries of tools for composition, determinization, minimization, best path (e.g. openFST)
- Semi-ring concept which allows symbols and weights to be generalized
- A complete ASR decoder could be written in a few lines of code (but no in practice)

Next steps

Break!

After break, overview of what it involves building an ASR system from scratch:

- Real use-case (Air Traffic Control Challenge)
- Including recent Deep Neural Networks architectures

Then you get a new dataset and create our own ASR system.

Part II

Building an ASR System: Case-Study

Part III

ASR System Building Lab

Preparation

Launching the recipe

Login to the master node then to a compute node:

```
ssh user@ec2-52-205-171-112.compute-1.amazonaws.com
qlogin -q all.q@g01
```

Create your directory and copy the recipe there:

```
cd /export/fs01
mkdir -p username/google_bengali
cd username/google_bengali
cp -R /export/fs01//jtrmal/google_bengali/s5 .
cd s5
```

Launch the recipe:

```
./run.sh --stage 1 |& tee run.log
```

Kaldi organization

```
s5
├── cmd.sh, path.sh, run.sh
├── conf: configuration files
├── local: scripts
├── steps: scripts
├── utils: scripts
├── corpus
└── data
    ├── dev
    ├── train
    ├── lang
    └── local
        └── lang
```

run.sh

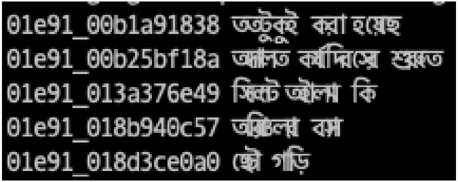
```
# Begin configuration section.
stage=0
corpus=./corpus
nj=4
dev_nj=6

# End configuration section
. ./utils/parse_options.sh

# initialization PATH
. ./path.sh  || die "File path.sh expected";
. ./cmd.sh  || die "File cmd.sh expected to exist"
```

```
if [ $stage -le 0 ]; then
  ./local/download_data.sh --datadir $corpus
fi

if [ $stage -le 1 ]; then
  echo "Preparing data and training language models"
  local/prepare_data.sh $corpus/
  local/prepare_dict.sh $corpus/
  utils/prepare_lang.sh data/local/dict "<UNK>" data/local/lang data/lang
  local/prepare_lm.sh $corpus/
fi
```

prepare_data.sh: text

```
utt_id WORD1 WORD2 WORD3 WORD4 ...
```

```
head -5 data/dev/text
```



prepare_data.sh:wav.scp

```
head -5 data/dev/wav.scp
```

```
01e91_00b1a91838 sox ./corpus/asr_bengali/data/00/00b1a91838.flac -t wav -r 16000 -|
01e91_00b25bf18a sox ./corpus/asr_bengali/data/00/00b25bf18a.flac -t wav -r 16000 -|
01e91_013a376e49 sox ./corpus/asr_bengali/data/01/013a376e49.flac -t wav -r 16000 -|
01e91_018b940c57 sox ./corpus/asr_bengali/data/01/018b940c57.flac -t wav -r 16000 -|
01e91_018d3ce0a0 sox ./corpus/asr_bengali/data/01/018d3ce0a0.flac -t wav -r 16000 -|
```

prepare_data.sh:utt2spk

```
head -5 data/dev/utt2spk
```

```
01e91_00b1a91838 01e91
01e91_00b25bf18a 01e91
01e91_013a376e49 01e91
01e91_018b940c57 01e91
01e91_018d3ce0a0 01e91
```

Training, dev, eval split

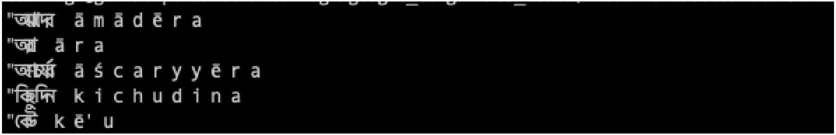
```
wc -l data/dev/*
   2790 data/dev/text
   2790 data/dev/utt2spk
     51 data/dev/spk2utt
   2790 data/dev/wav.scp

wc -l data/train/*
  12000 data/train/text
  12000 data/train/utt2spk
    457 data/train/spk2utt
  12000 data/train/wav.scp
```

prepare_dict.sh

Corpus-specific language.

```
lexicon.txt
nonsilence_phones.txt
optional_silence.txt
silence_phones.txt
oov.txt
```

```
head -5 ./data/local/dict/lexicon.txt
```



```
"আমাদের ā m ā d ē r a
"আর ā r a
"আশ্চর্য্যের ā ś c a r y y ē r a
"কিছুদিন k i c h u d i n a
"কেউ k ē ' u
```

data/lang

Script generated files.

```
L.fst
L_disambig.fst
oov.int
oov.txt
phones.txt
topo
words.txt
```

Language model training

prepare_lm.sh

```
--------------------
Computing perplexity
--------------------
data/srilm//6gram*: No such file or directory
data/srilm//6gram*
data/srilm//5gram.me.gz      file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -31870.21  ppl=  527.6197  ppl1=  3750.557
data/srilm//4gram.me.gz      file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -31889.62  ppl=  529.638   ppl1=  3769.404
data/srilm//3gram.me.gz      file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -32150.61  ppl=  557.5357  ppl1=  4032.195
data/srilm//3gram.kn111.gz   file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -32288.07  ppl=  572.8139  ppl1=  4177.881
data/srilm//3gram.kn011.gz   file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -32288.07  ppl=  572.8139  ppl1=  4177.881
data/srilm//4gram.kn0111.gz  file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -32511     ppl=  598.4898  ppl1=  4425.451
data/srilm//2gram.kn11.gz    file  data/srilm//dev.txt:  2790  sentences,  8917  words,  0  OOVs  0  zeroprobs,  logprob=  -32574.12  ppl=  605.9663  ppl1=  4498.174
```

format_lm.sh

```
data/lang_test:
    G.fst
    L.fst
    L_disambig.fst
data/lang_test_fg:
    G.carpa
    G.fst
```

path.sh, cmd.sh

path.sh sets up environment variables to point to Kaldi and tools installation directories.

```
export KALDI_ROOT=/export/fs01/jtrmal/kaldi
```

cmd.sh defines how parallelization is implemented.

- run.pl runs tasks on the local machine.
- queue.pl allocates jobs on a cluster using Sun Grid Engine.
- slurm.pl allocates jobs on a cluster using SLURM.

queue.pl and slurm.pl need to be configured with your cluster queue names.

HMM-GMM training

Feature Extraction

```
  # Feature extraction
  for x in train dev; do
      steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/$x exp/make_mfcc/$x mfcc
      steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x mfcc
  done
fi
```

Training steps

- Extract features: make_mfcc.sh
- Train monophones: train_mono.sh
- Align monophones: align_si.sh
- Train small triphones: train_deltas.sh
- Align small triphones: align_si.sh
- Train large triphones: train_deltas.sh
- Align large triphones: align_si.sh
- Train LDA+MLLT triphones: train_sat.sh

Decoding

In several places, evaluate word error rate with small LM (decode.sh) or large LM (lmrescore.sh).

```
(
echo "Decoding the dev set using SAT+FMLLR models."
utils/mkgraph.sh data/lang_test  exp/tri3b exp/tri3b/graph
steps/decode_fmllr.sh --nj $dev_nj --cmd "$decode_cmd" \
    exp/tri3b/graph  data/dev exp/tri3b/decode_dev

steps/lmrescore_const_arpa.sh  --cmd "$decode_cmd" \
    data/lang_test/ data/lang_test_fg/ data/dev \
    exp/tri3b/decode_dev exp/tri3b/decode_dev.rescored
echo "SAT+FMLLR decoding done."
) &
```

Debugging

Debugging

- Monitoring progress on cluster: `qstat`
- Restarting `--stage n`
- Scripts logs, cluster logs
- Cutting and pasting script lines

```
exp/make_unk/log
exp/mono_ali/log
exp/mono/log
exp/tri1_ali/log
exp/tri1/log
exp/tri2a_ali/log
exp/tri2a/log
exp/tri2b_ali/log
exp/tri2b/log
exp/tri3b_ali_train_sp/log
```

Looking at contents

Ark and scp files.

```
copy-feats scp:data/train/feats.scp ark,t:- | head -2
```

```
18a52_015f1ea678  [
  59.17816 -21.7955 2.345665 -2.779956 -8.278467 -12.64311 2.780146 3.892065 -7.968018 -3.988128 0.4810228 0.7078037 -8.235347
  60.1958 -21.35863 0.1444345 -5.106386 -3.468817 -9.695787 -9.42878 -7.550163 -9.84296 4.220254 -2.637221 -0.2019124 2.014682
  60.82346 -21.35863 -2.007355 -3.245242 -7.304714 -10.18701 -9.576305 -9.970974 -17.82411 -3.909159 -7.350846 -3.140995 -4.383737
  58.66934 -23.76137 -5.887454 -10.0153 -15.58162 -16.57288 -10.4843 -14.25034 -8.124263 -12.63851 -16.48274 -6.429967 -8.455023
```

Examples

`tree-info exp/tri3b/tree`

```
tree-info exp/tri3b/tree
num-pdfs 3298
context-width 3
central-position 1
```

`gmm-info exp/tri3b/final.mdl`

```
gmm-info exp/tri3b/final.mdl
number of phones 273
number of pdfs 3298
number of transition-ids 26178
number of transition-states 13069
feature dimension 40
number of gaussians 40065
```

Examples

```
steps/get_train_ctm.sh data/train data/lang exp/tri2b_ali
head exp/tri2b_ali/ctm
```

```
18a52_015f1ea678 1 0.950 0.480 জীন
18a52_015f1ea678 1 1.430 0.870 বৃত্তক
18a52_02f042f342 1 0.940 0.540 বেতন
18a52_02f042f342 1 1.480 0.570 কফিন
18a52_02f042f342 1 2.050 0.410 গঠন
18a52_02f042f342 1 2.460 0.280 করা
18a52_02f042f342 1 2.740 0.360 হয়
```

```
grep 18a52_015f1ea678 data/train/text
```

```
18a52_015f1ea678 জীন বৃত্তক
```

Error rate

```
find exp -name "best_wer" | xargs cat | sort -k2,2g
```

```
%WER 13.19 [ 1185 / 8986, 103 ins, 176 del, 906 sub ] exp/chain/tdnn_1c/decode_dev.rescored/wer_11_0.0
%WER 26.02 [ 2338 / 8986, 155 ins, 345 del, 1838 sub ] exp/chain/tdnn_1c/decode_dev/wer_10_0.0
%WER 28.91 [ 2598 / 8986, 203 ins, 479 del, 1916 sub ] exp/tri3b/decode_dev.rescored/wer_17_0.5
%WER 32.49 [ 2920 / 8986, 239 ins, 542 del, 2139 sub ] exp/tri2b/decode_dev.rescored/wer_15_0.0
%WER 32.94 [ 2960 / 8986, 214 ins, 583 del, 2163 sub ] exp/tri2a/decode_dev.rescored/wer_16_0.0
%WER 33.26 [ 2989 / 8986, 244 ins, 604 del, 2141 sub ] exp/tri1/decode_dev.rescored/wer_14_0.5
%WER 39.32 [ 3533 / 8986, 221 ins, 673 del, 2639 sub ] exp/tri3b/decode_dev/wer_17_1.0
%WER 45.19 [ 4061 / 8986, 229 ins, 797 del, 3035 sub ] exp/tri2b/decode_dev/wer_16_0.5
%WER 46.54 [ 4182 / 8986, 249 ins, 778 del, 3155 sub ] exp/tri2a/decode_dev/wer_16_0.0
%WER 46.76 [ 4202 / 8986, 274 ins, 784 del, 3144 sub ] exp/tri1/decode_dev/wer_16_0.0
%WER 47.51 [ 4269 / 8986, 338 ins, 709 del, 3222 sub ] exp/tri3b/decode_dev.si/wer_14_0.0
```

DNN Training

DNN Training

- i-vectors
- egs
- network configuration