ASR with Kaldi Tutorial

Gilles Boulianne¹  Vishwa Gupta¹  Jan Trmal²  Jérôme Labonté¹  Simon Desrochers¹
Presented at École de Technologie Supérieure, Montréal, June 10, 2019

¹Centre de recherche Informatique de Montréal

²Johns Hopkins University
<table>
<thead>
<tr>
<th>Time</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 AM to 10h20 AM</td>
<td>Fundamentals</td>
</tr>
<tr>
<td>10:20 AM to 10h40 AM</td>
<td><em>Break</em></td>
</tr>
<tr>
<td>10:40 AM to 11h30 AM</td>
<td>Case-study: Air Traffic Control Challenge</td>
</tr>
<tr>
<td>11:30 AM to 12h00 PM</td>
<td>Prepare and launch your own ASR</td>
</tr>
<tr>
<td>12:00 PM to 01:30 PM</td>
<td><em>Lunch Break</em></td>
</tr>
<tr>
<td>01:30 PM to 03:00 PM</td>
<td>ASR System Building Lab</td>
</tr>
<tr>
<td>03:00 PM to 03:30 PM</td>
<td><em>Coffee break</em></td>
</tr>
<tr>
<td>03:30 PM to 05:00 PM</td>
<td>Debugging and error analysis</td>
</tr>
</tbody>
</table>
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


- 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

From: Jan Trmal et al., "Kaldi ASR Tutorial for SLTU 2018"
Automatic Speech Recognition
ASR: From Sound to Computation

Sound in, computation, words out.

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
From spectrogram and images, feature vector representation.

Matrix \([\text{time} \times D]\):

\[
O = [o_1, o_2, \ldots, o_T] = o_t \quad \text{for} \quad t = 1, \ldots, T
\]

where \(o_t\) is a feature vector in \(\mathcal{R}^D\), one every 10 ms.
Maximum likelihood approach: not only one, but most successful.

Given:

- \( O \): a sequence of "observations" (feature vectors)
- \( P(W | O) \): the probability distribution of a word sequence given an observation sequence

\[
O = [o_1, o_2, \ldots, o_T]
\]

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

Find \( \hat{W} = \arg\max_W P(W | O) \)

Estimate \( P(W | O) \) directly: pattern recognition approach.

Not successful in the past, now revisited as end-to-end modeling.
Baye’s rule:

\[
P(W \mid O) = \frac{P(O \mid W)P(W)}{P(O)}
\]  

(1)

\[
\hat{W} = \arg\max_W P(O \mid W)P(W)
\]

(2)

for a given \(O\).

Called a generative model: \(W \rightarrow O\).

Sub-problems:

- Estimate model

- Find best \(W\) that maximizes \(P(O \mid W)\) and \(P(W)\).
$P(W)$ in ASR equation.

Chain rule allows the decomposition:

$$P(w_i, \ldots, w_1) = P(w_i | w_{i-1}, \ldots, w_1) \cdot P(w_{i-1}, \ldots, w_1) \quad (3)$$

$$P(s_i) = P(w_i | s_{i-1})P(s_{i-1}) \quad (4)$$

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

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

Useful because: compact representation, regular grammar.
$P(\text{the, big, cat}) = 1.0 \times 0.25 \times 1.0 = 0.25 \quad (5)$

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

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

$P(\text{a, cat}) = 1.0 \times 0.90 \times 1.0 = 0.90 \quad (8)$
**ASR: Acoustic Model**

\[ P(O | W) : \text{chain rule, and independence assumption.} \]

Boils down to:

\[
P(O | Q)P(Q) = \prod_{i=1}^{T} P(o_i | q_i) \times \prod_{i=1}^{T} P(q_i | q_{i-1})
\]

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

How do we compute \( P(o_t | q_i) \) ?
ASR: Gaussian Mixture Model

Gaussian distribution pdf:

\[ P(o_t) = \mathcal{N}(o_t; \mu, \sigma^2) \]

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

Mixture of Gaussian pdf:

\[ P(o_t) = \sum_{c=1}^{C} \phi_c \mathcal{N}(o_t; \mu_c, \sigma_c^2) \]

Parameters for each pdf: \( \phi_c, \mu_c, \sigma_c \) for \( c = 1, \ldots, C \)

Final form: \( P(o_t \mid pdf_i) \)
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.
Monophones: context-independent phones.

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

\[ W \, \text{AY} \, T \rightarrow W+\text{AY} \, W-\text{AY}+T \, \text{AY}+T \]

Decision tree: clusters several triphones sharing one pdf.

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

\[ P(O \mid W) = \sum_Q P(O \mid Q) P(Q \mid W) P(W) \]

And the best path:

\[ \hat{W} = \arg\max_w P(O \mid W) P(W) \]  \hspace{1cm} (9)

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

\[ P(W \mid O) = \frac{P(O \mid W) P(W)}{P(O)} \]  \hspace{1cm} (10)
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, ...
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
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.
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.
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.
So far, we have a word-based ASR model $w \rightarrow o$

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

Phonemes are easier to model and more flexible:

$$w \rightarrow p \rightarrow o$$

Coarticulation modeled with phonemes-in-context:

$$w \rightarrow p \rightarrow pic \rightarrow o$$

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

$$w \rightarrow p \rightarrow pic \rightarrow pdf \rightarrow o$$
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 $P(o_t \mid pdf_i)$

$$\hat{W} = \text{bestpath}(O \circ H \circ C \circ L \circ G)$$
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
# Begin configuration section.
stage=0
corpus=./corpus
nj=4
dev_nj=6

# End configuration section
./utils/pause_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
utt_id WORD1 WORD2 WORD3 WORD4 ...

head -5 data/dev/text
head -5 data/dev/wav.scp
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
```
Corpus-specific language.

lexicon.txt
nonsilence_phones.txt
optional_silence.txt
silence_phones.txt
oov.txt
head -5 ./data/local/dict/lexicon.txt
Script generated files.

L.fst
L_disambig.fst
oov.int
oov.txt
phones.txt
topo
words.txt
Language model training

prepare_lm.sh

format_lm.sh

data/lang_test:
  G.fst
  L.fst
  L_disambig.fst

data/lang_test_fg:
  G.carpa
  G.fst
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
In several places, evaluate word error rate with small LM (decode.sh) or large LM (lmrescore.sh).
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
```
Ark and scp files.

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

grep 18a52_015f1ea678 data/train/text
Error rate

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

```
%WER 13.19 [ 1185 / 8986, 103 ins, 176 del, 906 sub ] exp/ch.../1c/decode_dev.rescored/wer_11_0.0
%WER 26.02 [ 2338 / 8986, 155 ins, 345 del, 1838 sub ] exp/ch.../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