An Introduction to the Kaldi Speech Recognition Toolkit

Presenter: 高予真 2014.01.02

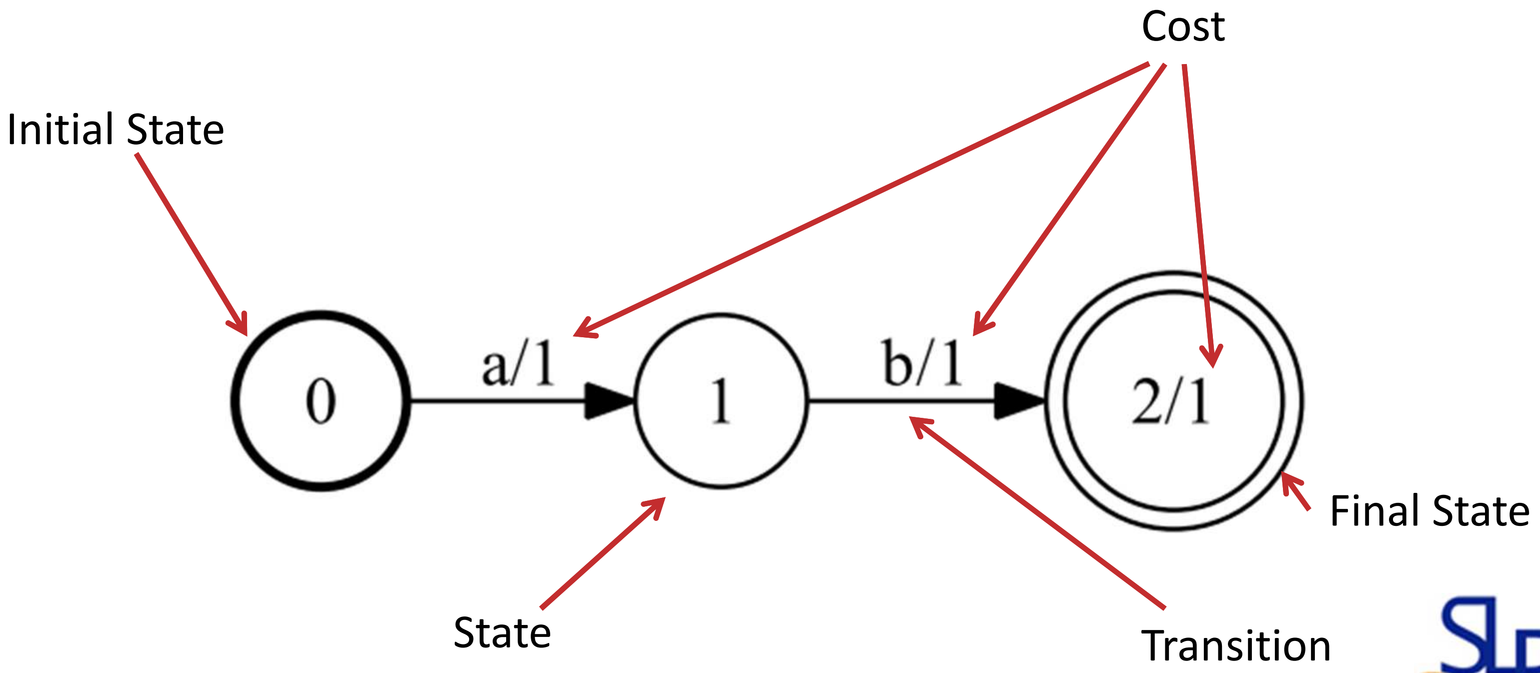
Outline

- A Brief Introduction to WFSTs
- The Kaldi Toolkit
- Overview of Kaldi Features
- A Simple Example
- Appendix

- A Brief Introduction to WFSTs
- The Kaldi Toolkit
- Overview of Kaldi Features
- A Simple Example
- Appendix

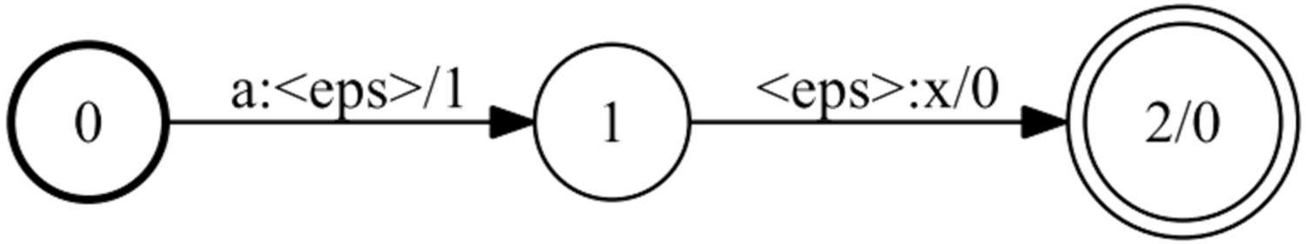
Weighted Finite State Acceptors

- Map "ab" to the total cost on the path; map all other strings to "zero"
- "Costs" are semirings



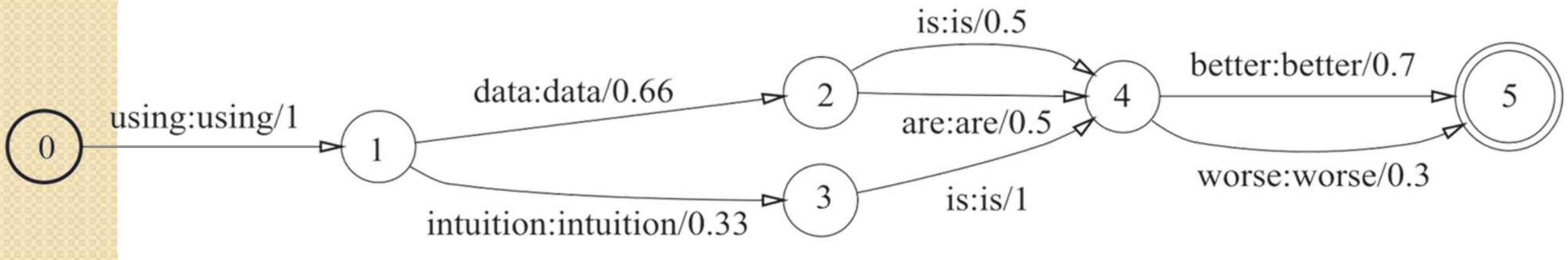
Weighted Finite State Transducers (WFSTs)

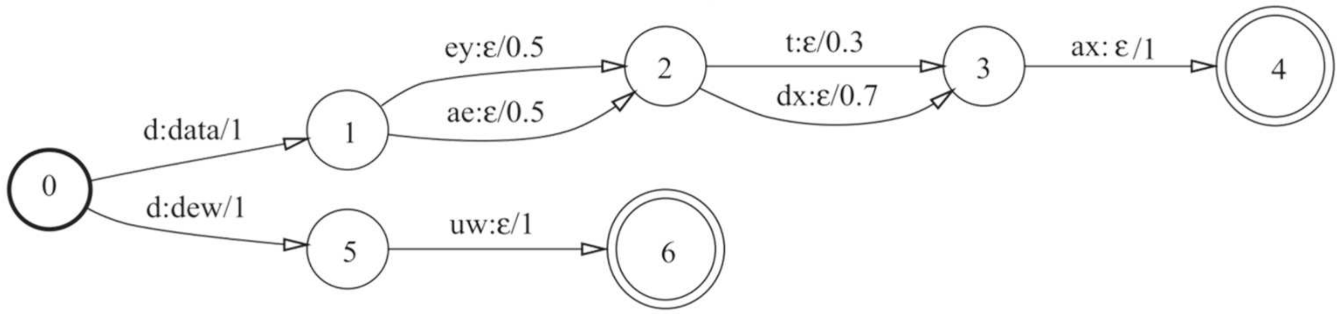
- Two labels on each edge
- Mapping (the “a”→”x” mapping) to cost 1



Why WFSTs?

- Efficient algorithms exists
- A unified framework to represent different layers of knowledge
- Can be optimized at training phase





WFSTs and ASR

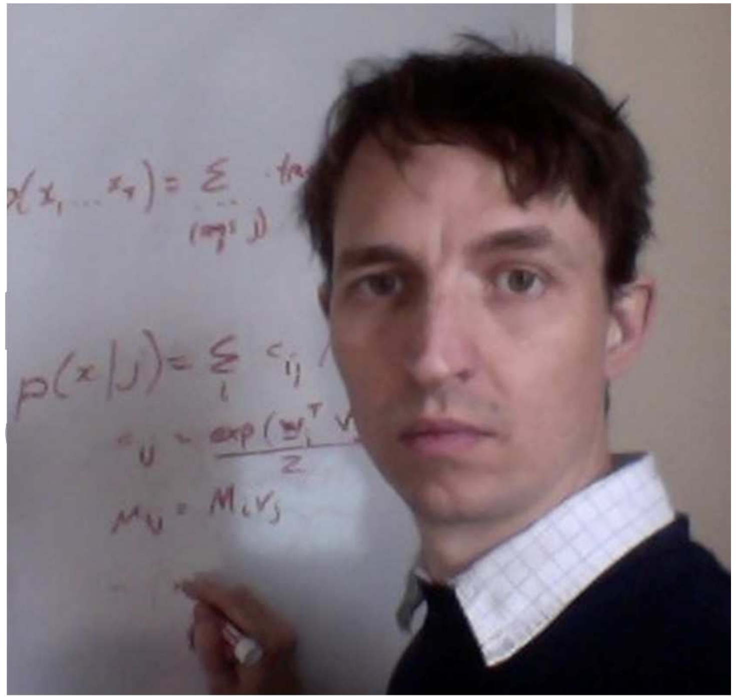
- The decoding graph: min(det($H \circ C \circ L \circ G$))

- H: mapping from PDFs to context labels
- C: mapping from context labels to phones
- L: mapping from phones to words
- G: grammar or language model

- A Brief Introduction to WFSTs
- The Kaldi Toolkit
- Overview of Kaldi Features
- A Simple Example
- Appendix

The Kaldi Toolkit

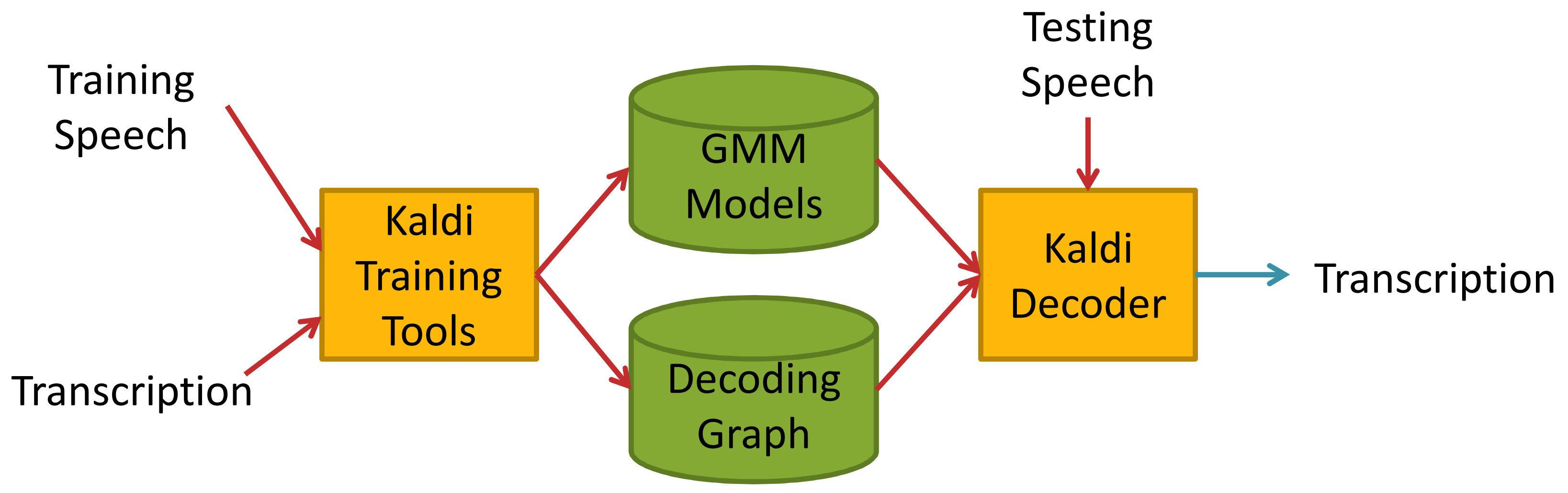
- Kaldi: an Ethiopian sl
  discovered the coffe



- A WFST-based speech recognition toolkit written mainly by Daniel Povey
- Initially born in a speech workshop in JHU in 2009, with some guys from Brno University of Technology

The Kaldi Toolkit

- Kaldi is specifically designed for speech recognition research application



Pros and Cons of using Kaldi

- Pros
  - Modular source, open license
  - Plenty of example scripts
  - Optimized for LVCSR tasks
  - Using pipes to significantly reduce disk I/O
- Cons
  - Commands and defaults change frequently
  - A little hard to work with on Windows
  - Almost impossible to use without some knowledge on shell scripting

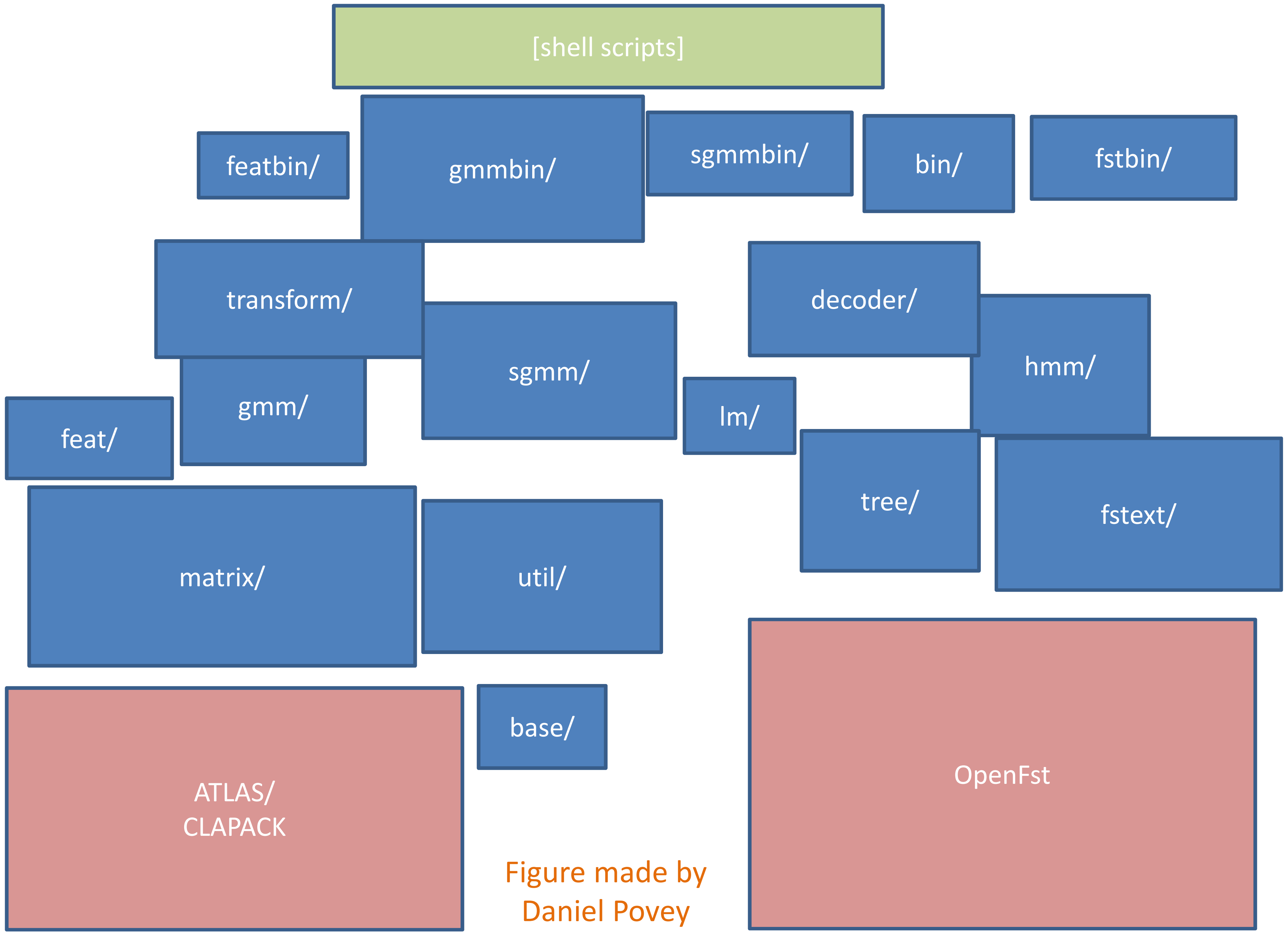


Figure made by
Daniel Povey

General Properties of Kaldi

- A C++ library of various speech tools
- The command-line tools are just thin wrappers of the underlying library

```
gmm-decode-faster --verbose=2 \
    --config=conf/file \
    --print-args=true \
    --acoustic-scale=0.09 \
    model.mdl \
    ark:decoding_graph.input \
    scp:feature.input \
    ark:text.output
```

Standard Arguments

Application-specific Arguments

Input/Output Files

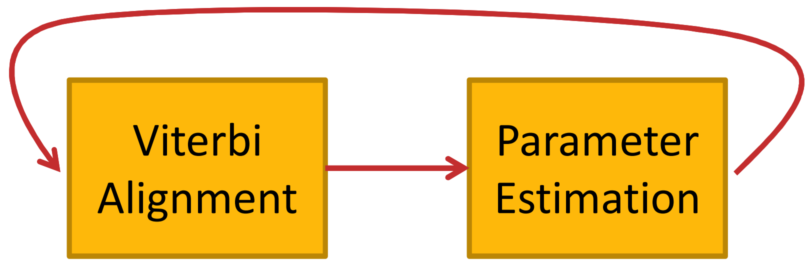
- A Brief Introduction to WFSTs
- The Kaldi Toolkit
- Overview of Kaldi Features
- A Simple Example
- Appendix

Feature Processing

- Basic MFCCs and PLPs
- Conventional Delta operations
- An unified framework for feature transformation
  - The “Transform” part does not know what the transform is at all
  - The “Estimate” part supports LDA, HLDA, fMLLR, MLLT, VTLN, etc.

Acoustic Modeling

- Standard maximum likelihood and MPE training of GMMs and subspace GMMs
- They don’t like the idea of “embedded training”, instead…



- “We don’t believe it’s better than Viterbi; and Viterbi makes it convenient to write alignments to disk.” – Daniel Povey

Acoustic Modeling

- Context-dependent acoustic modeling, with crazily wide context support (for example hepta-phone)

- Tree clustering according to:
  - Pre-defined questions
  - Data-driven clustering
  - Phone position and stress

Decoding

- Represents conventional HMM as a series of GMM and a transition graph, which is encoded in the decoding graph
- Decoding is done by just finding the Viterbi path in the decoding graph
- Three decoders available:
  - A simple decoder (for learning purpose)
  - A fast decoder (highly optimized and ugly)
  - An accurate decoder (very slow)

Decoding

- Decoders do not "know about" the HMM topology or GMMs, only a "decodable" interface, which has function that says "give me score for this"

- This make it easier to incorporate Kaldi with other kind of acoustic models

- A Brief Introduction to WFSTs
- The Kaldi Toolkit
- Overview of Kaldi Features
- A Simple Example
- Appendix

The Task

- MATBN corpus
  - Mandarin broadcast news
  - 34,672 training utterances
  - 292 testing utterances
- Use initial-final phone set
- Lexicon contains about 72,000 words
- Simple maximum-likelihood-trained GMM
- No context dependency

Data Preparation – File List

- Use the Kaldi Script File format (SCP)
- Name is a “standard token”, which means a string without any white space

```
matbn_tr_000033_000 34672_30100/MATBN_000033_000.wav
matbn_tr_000033_001 34672_30100/MATBN_000033_001.wav
matbn_tr_000033_002 34672_30100/MATBN_000033_002.wav
matbn_tr_000033_003 34672_30100/MATBN_000033_003.wav
matbn_tr_000033_004 34672_30100/MATBN_000033_004.wav
```

Name of this utterance

Wave file of this utterance

Data Preparation – Labels

- Use the Kaldi Archive File format (ARK)
- Most things in Kaldi are stored in this format, with right-hand side filled with different things

```
matbn_tr_000033_002 不過 司法院 認為 法院 的 庭長 只是 行政 兼職
matbn_tr_000033_003 免 兼 庭長 之後 仍然 是 相同 職 等 的 法官
matbn_tr_000033_004 職務 調動 只是 內部 的 行政 管理
matbn_tr_000033_005 當事人 不可以 表示 不服
```

Name of this utterance

Transcription of this utterance

Data Preparation – Lexicon

- Also the Kaldi Archive File format (ARK)
- Represent words in initial-final phone set
- Be sure to use UTF-8 encoding, things might blow up on Big5

```
三義 s en sic i
三聯單 s en l ian d en
三胞胎 s en b au t ai
三腳架 s en ji iau ji ia
三腳貓 s en ji iau m au
三芝 s en j empt1
三藩市 s en f en sh empt1
三角 s en ji iau
```

Data Preparation – ID Mapping

- Kaldi does not deal with symbols, only word IDs and phone IDs

Null Symbol

```
<eps> 0
sil 1
a 2
ai 3
…
ueng 60
uo 61
#0 62
#1 63
#2 64
#3 65
```

```
<eps> 0
一 1
一一 2
一丁點 3
一下 4
一丘之貉 5
一乾二淨 6
一五一十 7
一些 8
…
齰 71694
#0 71695
```

Disambiguation Symbols

Data Preparation – HMM Topology

```
<Topology>
<TopologyEntry>
<ForPhones>
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 48 49 50 51 52 53 54 55 56 57 58
59 60 61
</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 <PdfClass> 3 <Transition> 3 0.75 <Transition> 4 0.25 </State>
<State> 4 </State>
</TopologyEntry>
<TopologyEntry>
<ForPhones>
1
</ForPhones>
<State> 0 <PdfClass> 0 <Transition> 0 0.25 <Transition> 1 0.25 <Transition> 2 0.25
<Transition> 3 0.25 </State>
<State> 1 <PdfClass> 1 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25
<Transition> 4 0.25 </State>
<State> 2 <PdfClass> 2 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25
<Transition> 4 0.25 </State>
<State> 3 <PdfClass> 3 <Transition> 1 0.25 <Transition> 2 0.25 <Transition> 3 0.25
<Transition> 4 0.25 </State>
<State> 4 <PdfClass> 4 <Transition> 4 0.75 <Transition> 5 0.25 </State>
<State> 5 </State>
</TopologyEntry>
</Topology>
```

The Disambiguation Symbol

- Homophones will cause some problems in the composing of the decoding graph
- Need to add an unique empty symbol for each homophones
- There is a script in the Kaldi example folder to do this automatically

```
add_lex_disambig.pl lexicon lexicon_disambig
```

Command Filenames

Construct Graphs for Lexicon and LM

```
make_lexicon_fst.pl lexicon_disambig 0.5 sil | \
  fstcompile --isymbols=map_phone --osymbols=map_word \
  --keep_isymbols=false --keep_osymbols=false | \
  fstaddselfloops 62 71695 | \
  fstarcsort --sort_type=olabel > L.fst
```

```
arpa2fst --natural-base=false language_model | \
  fstprint | eps2disambig.pl | s2eps.pl | \
  fstcompile –isymbols=map_word --osymbols=map_word \
  --keep_isymbols=false --keep_osymbols=false | \
  fstrmepsilon > G.fst
```

Feature Extraction

```
compute-mfcc-feats scp:filelist_wave ark:archive_mfcc

compute-cmvn-stats ark:archive_mfcc ark:stats_cmvn

apply-cmvn ark:stats_cmvn ark:archive_mfcc ark:archive_cmvn

add-deltas --delta-order=2 --delta-window=2 \
  ark:archive_cmvn ark:archive_cmvn+delta
```

- Most of the time, we process training data and testing data separately, for easier access

Initialize Models

- Construct a basic GMM for each phone in a similar way to HCompV from HTK
- Also, compose the training transcription with lexicon to make training faster

```
gmm-init-mono --train-feats=ark:feature_train \
  topo 39 model_init.mdl tree

compile-train-graphs tree model_init.mdl \
  L.fst ark:label_train ark:train.fsts
```

Training GMMs – First Iteration

- We don’t have a valid alignment now
- For the first iteration of GMM training, an equal alignment is used

```
align-equal-compiled ark:train.fsts ark:feature_train \
  ark:- | \
  gmm-acc-stats-ali --binary=true model_init.mdl \
  ark:feature_train ark:- accumulator

gmm-est --min-gaussian-occupancy=3 \
  model_init.mdl "gmm-sum-accs - accumulator|" \
  model_1.mdl
```

Note the use of pipe
inside a parameter

Training GMM – Increasing Mixtures

- Increase the number of mixture inside each GMM to model that phone better
- Number of mixtures decided by algorithm

```
gmm-boost-silence --boost=0.75 1 model_1.mdl model_1_ali.mdl

gmm-align-compiled --transition-scale=1.0 --acoustic-scale=0.1 \
  --self-loop-scale=0.1 --beam=10 --retry-beam=40 \
  model_1_ali.mdl ark:train.fsts ark:feature_train ark:- | \
  gmm-acc-stats-ali --binary=true model_1.mdl \
  ark:feature_train ark:- accumulator

gmm-est --min-gaussian-occupancy=3 --mix-up=200 --power=0.25 \
  model_init.mdl "gmm-sum-accs - accumulator|" \
  model_2.mdl
```

Construct the Decoding Graph

```
fsttablecompose L.fst G.fst | \
  fstdeterminizestar --use-log=true | fstminimizeencoded > LG.fst

fstcomposecontext --context-size=1 --central-position=0 \
  --read-disambig-syms=list_disambig \
  --write-disambig-syms=ilabels_disambig \
  ilabels_tr LG.fst > CLG.fst

make-h-transducer --disambig-syms-out=tid_disambig \
  ilabels_tr tree model_final.mdl > H.fst

fsttablecompose H.fst CLG.fst | \
  fstdeterminizestar --use-log=true | \
  fstrmsymbols tid_disambig | fstrmepslocal | fstminimizeencoded | \
  add-self-loops --self-loop-scale=0.1 --reorder=true \
  model_final.mdl > HCLG.fst
```

Decoding

- The decoding stage is fairly fast and simple, since most hard work is done in the graph construction stage
- The int2sym.pl is also taken from the example scripts folder

```
gmm-decode-faster --acoustic-scale=0.09 --max-active=2500 \
  model_final.mdl HCLG.fst ark:feature_testing ark,t:- | \
  int2sym.pl -f 2- map_word > transcription_testing

compute-wer --text --mode=“strict" \
  ark:label_testing ark:transcription_testing
```

Final Notes

- Every researcher have different taste on recipe writing
- This recipe is specifically designed for MATBN, and is not the only way to do it
- You will see more distinct styles from Kaldi’s example scripts for TI-DIGITS, WSJ, VoxForge, Switchboard-1, etc.

- A Brief Introduction to WFSTs
- The Kaldi Toolkit
- Overview of Kaldi Features
- A Simple Example
- Appendix

Installing Cygwin

- See SRILM slides for detailed instructions
- You will need to install some additional packages:
  - Make
  - Patch
  - Ed
  - Subversion
  - Automake
  - Liblapack0
  - Gcc-core
  - Gcc-g++
  - Gcc-fortran

Obtaining Kaldi

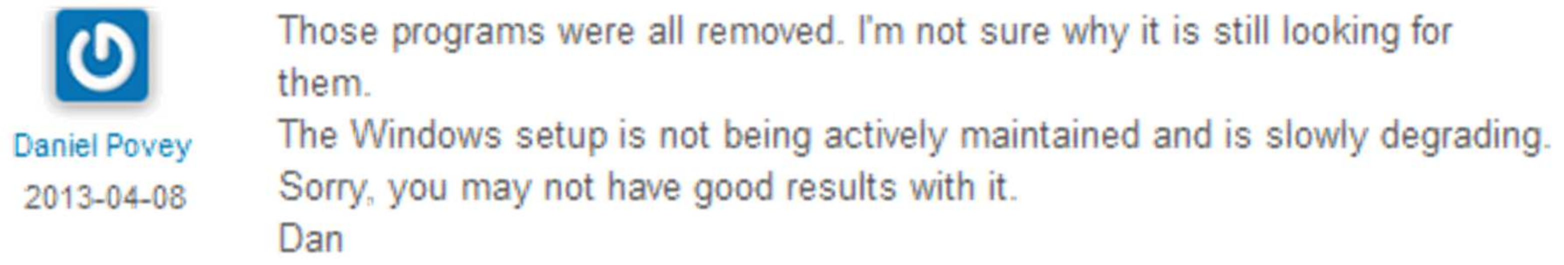
- Under the cygwin command line or linux terminal …

```
svn co svn://svn.code.sf.net/p/kaldi/code/trunk kaldi-trunk
```

- Then follow the instructions in a file called “INSTALL”
- It’s normal to take several hours to compile the underlying matrix library and Kaldi itself

Caveats of Kaldi on Windows

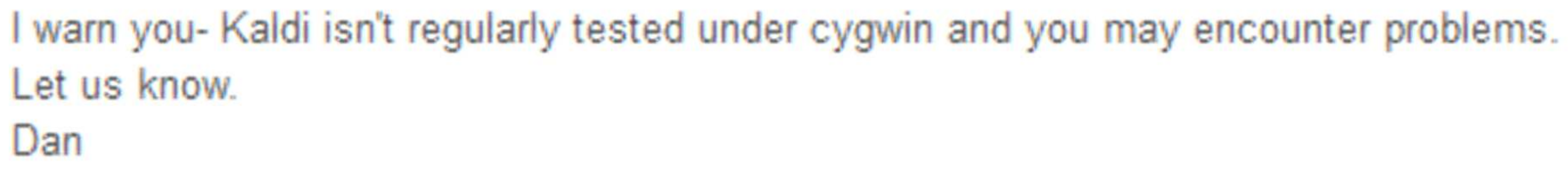
- Visual studio simply don't work



Daniel Povey
2013-04-08

Those programs were all removed. I'm not sure why it is still looking for them.
The Windows setup is not being actively maintained and is slowly degrading.
Sorry, you may not have good results with it.
Dan

- Cygwin is needed, even though……



I warn you- Kaldi isn't regularly tested under cygwin and you may encounter problems.
Let us know.
Dan

- … and cygwin is a big minefield, if a version works for you, do not attempt to update it

Suggestions

- Keep the source code after compilation, you will need them since there is no command documentation (like HTK Book)
- Check the command usage before using it (by run it with --help flag): things in the example scripts may be obsolete

Thank You!