KALDI

A toolkit for speech recognition research

(According to legend, Kaldi was the Ethiopian goatherd who discovered the coffee plant).

Key aspects of the project

- Apache v2.0 license (very free)
- Available on Sourceforge
- Open source, collaborative project (we welcome new participants)
- C++ toolkit (compiles on Windows and common UNIX platforms)
- Has documentation and example scripts

KALDI

Overview of features

- Context-dependent LVCSR system (arbitrary phonetic-context width)
- FST-based training and decoding (we use OpenFst)
- Maximum Likelihood training
  - Working on lattice generation + DT.
- All kinds of linear and affine transforms
- Example scripts demonstrate VTLN, SAT, etc.

KALDI

Advantages versus other toolkits*

- Clean code, modular and extensible design
- Intended to be easy to understand and modify
- Very open license (Apache 2.0)
- Example scripts and documentation available
- Trying to build helpful and active community
- Good linear algebra support; FSTs
- Very scalable
- We intend to implement all state-of-the-art methods (inc. discriminative training)

*Disclaimer: some toolkits may have at least some of these advantages.

KALDI

Features not on current "to-do" list

- No on-line decoder (batch mode only)
  - It's mostly for speech-recognition research
- No explicit support for parallelization (MapReduce, MPI)
  - Would be too platform-specific… we use a HTK-like approach where you can sum up accumulator files.
- No scripting-language wrapper
  - Would force users to learn e.g. Python; we support configurability in different ways.
- No forward-backward training
  - We don't believe it's better than Viterbi; and Viterbi makes it convenient to write alignments to disk.

History of Kaldi (1/2)

- JHU 2009 workshop*, working (mostly) on Subspace Gaussian Mixture Models (SGMMs)
- Guys from Brno University of Technology (BUT) created "proto-Kaldi"…
    - Had FST-based decoding and SGMM training setup
    - Dependent on HTK for feature generation and building an initial GMM-based system
    - Entire solution was complex due to merging of two different setups.

*This workshop was funded by NSF Grant #IIS-0833652, with supplemental funding from Google, DARPA's GALE program, and JHU's HLTCoE. BUT researchers were partially supported in this period by Czech Ministry of Trade and Commerce Project # FR-TI1/034, Grant Agency of Czech Republic project no. 102/08/0707, and Czech Ministry of Education project no. MSM0021630528. Arnab Ghoshal was partially supported during this period by the European Community's Seventh Framework Programme under grant agreement number 213850 (SCALE).

History of Kaldi (2/2)

- In summer of 2010, some of us (+ new participants) went back to Brno for 2 months ("Kaldi workshop 2010"), hosted by Brno University of Technology.
- Aimed to build a self-contained, clean toolkit with no HTK dependency.
- Immediate goal was to create clean, releasable SGMM recipe.
- Wider goal of making a clean speech-recognition toolkit.
- Completed a lot of it that summer but not ready for release until last week.

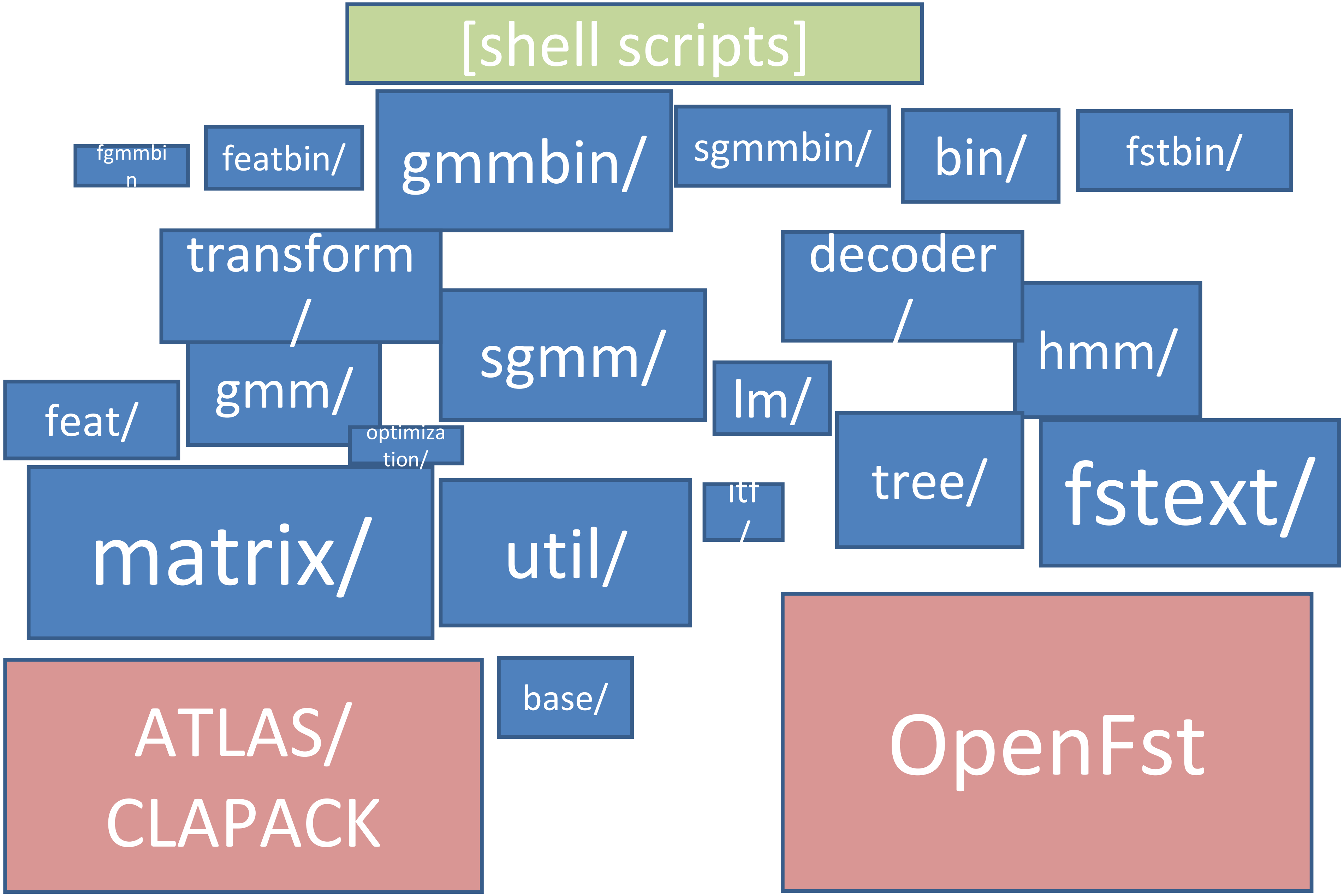
KALDI

Kaldi contributors

- Individuals who wrote code for Kaldi* so far:
  - Mohit Agarwal, Sandeep Boda[1], Gilles Boulianne, Lukas Burget, Arnab Ghoshal, Mirko Hannemann, Ondrej Glembek, Nagendra Goel[1], Pavel Matejka[2], Petr Motlicek, Daniel Povey[3], Yanmin Qian, Ariya Rastrow, Sandeep Reddy[1], Petr Schwarz[2], Jan Silovsky, Georg Stemmer, Karel Vesely, Haihua Xu.
- Also thanks to (non-exclusively)[++]:
  - Alex Acero, Pinar Akyazi, Honza Cernocky, Paul Dixon, JHU's CLSP staff + faculty, Tomas Kasparek, Renata Kohlova, Rico Malvar, Patrick Nguyen, Mike Riley, Rick Rose, Samuel Thomas, Geoffrey Zweig.

[1] GoVivace, Inc. [2] Phonexia s.r.o. [3] Microsoft Corp. (code contributed as employee)
*I.e. specifically for Kaldi [++] There are probably inadvertent oversights.

Kaldi dependency structure (approx)



Matrix library

- C++ wrapper for BLAS and CLAPACK linear algebra libraries (plus some extra code).
- Can use either (BLAS+CLAPACK), or ATLAS, or MKL, as external library.
- Supports generic, packed symmetric and packed triangular matrix formats.
- Supplies typical linear-algebra functionality (SVD, etc.), and FFT.
- Reusable: independent of rest of Kaldi code (except one small directory “base/”).

KALDI

OpenFst and fstext/

- OpenFst is open-source FST library (mostly from Google)
- We compile against it, e.g. decoding-graph object is an OpenFst object.
- fstext/ contains various extensions to OpenFst
  - E.g. implementation of on-demand context-dependency transducer
  - Our FST recipe is a little bit different from the standard one and requires slightly different FST algorithms (e.g. determinization with epsilon removal)

Kaldi I/O

- Based on C++ streams
- Supports binary and text-mode formats
- extended filenames: “-”, “gunzip –c foo.gz|”, “/offset/into/file:12345”
- Archive format: generic mechanism to index objects by strings (typically utterance id)

Tree building and clustering code

- Very generic clustering and tree building mechanisms
- Easy to build trees in various different ways (globally shared tree roots, etc.)
- Our current recipes use automatically generated questions (minimize hassle)
- Mechanisms scalable to wide context (e.g. quinphone) and large phone-sets
- In WSJ recipe we, in effect, ask questions about phone-position and stress (via expanded phone set and specially constrained questions… this is mostly set up at the script level)

HMM and transition modeling

- This code is separate from the “GMM” side of things (just treat states as integer ids)
- Can specify a “prototype” topology for each phone
- Transition is separately estimated depending on the p.d.f. index on the state it comes out of
- Mechanisms for turning these HMMs into FSTs
- In our FSTs, the (input) labels encode more information than just the p.d.f. index (e.g. encodes the phone, the position in the HMM)
- This is so we can train the transitions (and can work out the phone sequences from this index sequence)

Decoding-graph creation

- There is a C++ mechanism for creating decoding graphs (FSTs) in training time, from transcriptions
- These graphs are typically cached on disk
- We train using the Viterbi path through these graphs (redo Viterbi every few iterations)
- For the larger decoding graphs used in test time, we put relatively simple command-line tools together with a shell script
- Some of these are OpenFst tools, but mostly our own (using C++-level OpenFst mechanisms)

Gaussian Mixture Models (GMMs)

- Code for GMMs is fairly simple and passive
- Have avoided complex frameworks
- Representation of a single GMM
  - Likelihood evaluation; mutators
  - Separate class for accumulation and training
- Class for a collection of GMMs (indexed by integer pdf-index)… similar to vector<Gmm>
  - Corresponding “accumulator” class
- GMM code does not “know about” HMMs, transition models, linear transforms, etc.

Linear transform code

- Code for estimation of various linear transforms
  - LDA, HLDA, fMLLR/CMLLR, MLLT/STC, linear VTLN, "exponential transform" (something new, like VTLN), MLLR
  - This code is specifically for GMMs (would code these algorithms separately for other models)
- Linear transforms applied in a unified way (code does not "know" how they were estimated)
  - Usually applied as part of a pipe
- Mechanisms for regression trees for (fMLLR, MLLR)
  - Used in separate command-line decoders (don't want to complicate code that isn't doing this)

Decoders

- Decoders (currently) use fully expanded FSTs
- Currently 3 decoders on spectrum simple ←→fast
- But >3 command-line decoding programs!
- Decoders don't "know about" GMMs, HMMs, etc: just the FSTs, and "Decodable" interface
- "Decodable" interface has function that says "give me score for this (frame, index)"… like matrix lookup
- We "wrap" GMMs etc. in a thin wrapper that satisfies "Decodable" interface
- Command-line decoding programs always do one pass of decoding and are for a specific (decoder, model type).
- Multiple decoding passes done at script level (invoke decoder multiple times)

KALDI

Feature processing

- Support standard MFCC and PLP features
- A reasonable range of configurability (#mel bins, etc.)
- Read only .wav format
- Use external programs for format conversion, e.g. from sphere
- Typically write features (like other objects) all to a very large file
- Expansion with deltas, fMLLR, etc. typically done using pipes, on-the-fly, to minimize disk I/O
- In next talk will explain the framework for this.

Command-line tools

- Large number of command-line tools (>150), each with a fairly simple function
- Command-line tools take options e.g.

```
compute-mfcc-feats --use-energy=false  \
    ark:data/train_wav.scp  \
    ark,scp:data/train.ark,train,scp
```

- We rarely need to supply more than a few options to any given program
- Command line tools generally have quite simple code.
- C++ code doesn’t have to worry much about I/O (handled through templated code via “Table” concept… will explain after the break).

Scripts (example fragment)

```
#!/bin/bash
…
while [ $x -lt $numiters ]; do
 if echo $mllt_iters | grep -w $x >/dev/null; then # Do MLLT update.
    ( ali-to-post ark:$dir/cur.ali ark:- | \
       weight-silence-post 0.0 $silphonelist $dir/$x.mdl ark:- ark:- | \
       gmm-acc-mllt --binary=false $dir/$x.mdl "$featsub" ark:- $dir/$x.macc ) \
          2> $dir/macc.$x.log  || exit 1;

     est-mllt $dir/$x.mat.new $dir/$x.macc 2> $dir/mupdate.$x.log || exit 1;
     gmm-transform-means --binary=false $dir/$x.mat.new $dir/$x.mdl $dir/$[$x+1].mdl \
     2> $dir/transform_means.$x.log || exit 1;
     compose-transforms --print-args=false $dir/$x.mat.new $cur_lda $dir/$x.mat || exit 1;
     cur_lda=$dir/$x.mat

     feats="ark:splice-feats scp:data/train.scp ark:- | transform-feats $cur_lda ark:- ark:-|"
     # Subset of features used to train MLLT transforms.
     featsub="ark:scripts/subset_scp.pl 800 data/train.scp | splice-feats scp:- ark:- |
             transform-feats $cur_lda ark:- ark:-|"
 else
  ….
```

Scripts (points to note)

- Scripts quite complex
- Much of the configurability of Kaldi takes place at shell-script level
- This helps keep the C++ code simple
- Note use of pipes: features, alignments etc. are passed through pipes.

Selected results (WSJ)

%WER	Nov’92	Nov’93
Reichl (2000)	11.9	15.4
HTK (gender dep.) (ICASSP’94)	11.1	14.5
Kaldi	11.8	15.0

- SI-284 training, Sennheiser microphone, 20k open vocabulary test, bigram LM supplied with WSJ.
- Unadapted, cross-word triphones (but HTK system was gender-dependent)
- This is not our best result, just showing that with comparable algorithms we get comparable results)

KALDI

Speed, decoding issues (WSJ)

- We can’t yet decode with full trigram LM from WSJ (graph too large)… but pruned one is OK
- Working on this issue
- Decoding speed for previous results is about 0.5xRT (i.e. twice faster than real time)
- Training time: takes a few hours to train the previous system, on a single machine (using up to 3 CPUs)

Further results (RM)

%WER	Feb'89	Oct'89	Feb'91	Sep'92	Avg
HTK	2.77	4.02	3.30	6.29	4.10
Kaldi	3.20	4.10	2.86	6.06	4.06

- Both systems cross-word triphone with cepstral mean normalization
- HTK results from ICASSP'99 paper (Povey et. al)
  - Probably slightly better than RMHTK recipe due to variable #gauss per state.
- Decoding speed ~0.1xRT

RM: unadapted experiments

%WER	None
MFCC+Δ+ΔΔ	4.59
+mean normalization	4.16
MFCC+Δ+ΔΔ + MLLT/STC	4.59
Splice-9 + LDA	5.04
Splice-9 + LDA + MLLT/STC	4.35
Splice-9 + HLDA	4.61
Triple-deltas + HLDA	4.32
Triple-deltas + LDA + MLLT/STC	3.95
SGMM	3.15

- All results averaged over 6 RM test sets

RM: adapted experiments

%WER	Utt	Spk
MFCC+Δ+ΔΔ (fMLLR)	4.56	3.67
MFCC+Δ+ΔΔ + ET	3.35	3.32
MFCC+Δ+ΔΔ + VTLN	3.94	3.56
Splice-9 + LDA + ET	3.29	3.08
+ fMLLR		2.73
Splice-9 + LDA + MLLT/STC + SAT	5.10	2.75
SGMM + spk-vecs	2.72	2.68
SGMM + spk-vecs + fMLLR		2.53
SGMM + fMLLR		2.77

- “ET”= Exponential Transform (like VTLN)

Why use Kaldi

- Easy to use (once you learn the basics, and assuming you understand the underlying science)
- Easy to extend and modify.
- Redistributable: unrestrictive license, community project.
- If you stuff works or is interesting, the Kaldi team is open to including it and your example scripts in our central repository → more citations, as others build on it.