The Kaldi Speech Recognition Toolkit

Daniel Povey[1], Arnab Ghoshal[2],
Gilles Boulianne[3], Lukáš Burget[4,5], Ondřej Glembek[4], Nagendra Goel[6], Mirko Hannemann[4],
Petr Motlíček[7], Yanmin Qian[8], Petr Schwarz[4], Jan Silovský[9], Georg Stemmer[10], Karel Veselý[4]

[1] *Microsoft Research, USA,* dpovey@microsoft.com*;*
[2] *Saarland University, Germany,* aghoshal@lsv.uni-saarland.de*;*
[3] *Centre de Recherche Informatique de Montréal, Canada;* [4] *Brno University of Technology, Czech Republic;*
[5] *SRI International, USA;* [6] *Go-Vivace Inc., USA;* [7] *IDIAP Research Institute, Switzerland;* [8] *Tsinghua University, China;*
[9] *Technical University of Liberec, Czech Republic;* [10] *University of Erlangen-Nuremberg, Germany*

***Abstract*—We describe the design of Kaldi, a free, open-source toolkit for speech recognition research. Kaldi provides a speech recognition system based on finite-state transducers (using the freely available OpenFst), together with detailed documentation and scripts for building complete recognition systems. Kaldi is written is C++, and the core library supports modeling of arbitrary phonetic-context sizes, acoustic modeling with subspace Gaussian mixture models (SGMM) as well as standard Gaussian mixture models, together with all commonly used linear and affine transforms. Kaldi is released under the Apache License v2.0, which is highly nonrestrictive, making it suitable for a wide community of users.**

I. Introduction

Kaldi[1] is an open-source toolkit for speech recognition written in C++ and licensed under the Apache License v2.0. The goal of Kaldi is to have modern and flexible code that is easy to understand, modify and extend. Kaldi is available on SourceForge (see http://kaldi.sf.net/). The tools compile on the commonly used Unix-like systems and on Microsoft Windows.

Researchers on automatic speech recognition (ASR) have several potential choices of open-source toolkits for building a recognition system. Notable among these are: HTK [1], Julius [2] (both written in C), Sphinx-4 [3] (written in Java), and the RWTH ASR toolkit [4] (written in C++). Yet, our specific requirements—a finite-state transducer (FST) based framework, extensive linear algebra support, and a non-restrictive license—led to the development of Kaldi. Important features of Kaldi include:

Integration with Finite State Transducers: We compile against the OpenFst toolkit [5] (using it as a library).

Extensive linear algebra support: We include a matrix library that wraps standard BLAS and LAPACK routines.

Extensible design: We attempt to provide our algorithms in the most generic form possible. For instance, our decoders work with an interface that provides a score for a particular frame and FST input symbol. Thus the decoder could work from any suitable source of scores.

Open license: The code is licensed under Apache v2.0, which is one of the least restrictive licenses available.

Complete recipes: We make available complete recipes for building speech recognition systems, that work from widely available databases such as those provided by the Linguistic Data Consortium (LDC).

Thorough testing: The goal is for all or nearly all the code to have corresponding test routines.

The main intended use for Kaldi is acoustic modeling research; thus, we view the closest competitors as being HTK and the RWTH ASR toolkit (RASR). The chief advantage versus HTK is modern, flexible, cleanly structured code and better WFST and math support; also, our license terms are more open than either HTK or RASR.

The paper is organized as follows: we start by describing the structure of the code and design choices (section II). This is followed by describing the individual components of a speech recognition system that the toolkit supports: feature extraction (section III), acoustic modeling (section IV), phonetic decision trees (section V), language modeling (section VI), and decoders (section VIII). Finally, we provide some benchmarking results in section IX.

II. Overview of the toolkit

We give a schematic overview of the Kaldi toolkit in figure 1. The toolkit depends on two external libraries that are also freely available: one is OpenFst [5] for the finite-state framework, and the other is numerical algebra libraries. We use the standard "Basic Linear Algebra Subroutines" (BLAS)and "Linear Algebra PACKage" (LAPACK)[2] routines for the latter.

The library modules can be grouped into two distinct halves, each depending on only one of the external libraries (c.f. Figure 1). A single module, the `DecodableInterface` (section VIII), bridges these two halves.

Access to the library functionalities is provided through command-line tools written in C++, which are then called from a scripting language for building and running a speech recognizer. Each tool has very specific functionality with a small set of command line arguments: for example, there are separate executables for accumulating statistics, summing accumulators, and updating a GMM-based acoustic model

[1] According to legend, Kaldi was the Ethiopian goatherd who discovered the coffee plant.

[2] Available from: http://www.netlib.org/blas/ and http://www.netlib.org/lapack/ respectively.

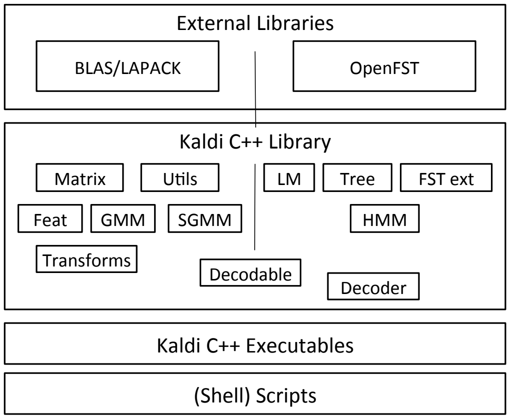

Fig. 1. A simplified view of the different components of Kaldi. The library modules can be grouped into those that depend on linear algebra libraries and those that depend on OpenFst. The *decodable* class bridges these two halves. Modules that are lower down in the schematic depend on one or more modules that are higher up.

using maximum likelihood estimation. Moreover, all the tools can read from and write to pipes which makes it easy to chain together different tools.

To avoid "code rot", We have tried to structure the toolkit in such a way that implementing a new feature will generally involve adding new code and command-line tools rather than modifying existing ones.

III. Feature Extraction

Our feature extraction and waveform-reading code aims to create standard MFCC and PLP features, setting reasonable defaults but leaving available the options that people are most likely to want to tweak (for example, the number of mel bins, minimum and maximum frequency cutoffs, etc.). We support most commonly used feature extraction approaches: e.g. VTLN, cepstral mean and variance normalization, LDA, STC/MLLT, HLDA, and so on.

IV. Acoustic Modeling

Our aim is for Kaldi to support conventional models (i.e. diagonal GMMs) and Subspace Gaussian Mixture Models (SGMMs), but to also be easily extensible to new kinds of model.

A. Gaussian mixture models

We support GMMs with diagonal and full covariance structures. Rather than representing individual Gaussian densities separately, we directly implement a GMM class that is parametrized by the *natural parameters*, i.e. means times inverse covariances and inverse covariances. The GMM classes also store the *constant* term in likelihood computation, which consist of all the terms that do not depend on the data vector. Such an implementation is suitable for efficient log-likelihood computation with simple dot-products.

B. GMM-based acoustic model

The "acoustic model" class `AmDiagGmm` represents a collection of `DiagGmm` objects, indexed by "pdf-ids" that correspond to context-dependent HMM states. This class does not represent any HMM structure, but just a collection of densities (i.e. GMMs). There are separate classes that represent the HMM structure, principally the topology and transition-modeling code and the code responsible for compiling decoding graphs, which provide a mapping between the HMM states and the pdf index of the acoustic model class. Speaker adaptation and other linear transforms like maximum likelihood linear transform (MLLT) [6] or semi-tied covariance (STC) [7] are implemented by separate classes.

C. HMM Topology

It is possible in Kaldi to separately specify the HMM topology for each context-independent phone. The topology format allows nonemitting states, and allows the user to pre-specify tying of the p.d.f.'s in different HMM states.

D. Speaker adaptation

We support both model-space adaptation using maximum likelihood linear regression (MLLR) [8] and feature-space adaptation using feature-space MLLR (fMLLR), also known as constrained MLLR [9]. For both MLLR and fMLLR, multiple transforms can be estimated using a regression tree [10]. When a single fMLLR transform is needed, it can be used as an additional processing step in the feature pipeline. The toolkit also supports speaker normalization using a linear approximation to VTLN, similar to [11], or conventional feature-level VTLN, or a more generic approach for gender normalization which we call the "exponential transform" [12]. Both fMLLR and VTLN can be used for speaker adaptive training (SAT) of the acoustic models.

E. Subspace Gaussian Mixture Models

For subspace Gaussian mixture models (SGMMs), the toolkit provides an implementation of the approach described in [13]. There is a single class `AmSgmm` that represents a whole collection of pdf's; unlike the GMM case there is no class that represents a single pdf of the SGMM. Similar to the GMM case, however, separate classes handle model estimation and speaker adaptation using fMLLR.

V. Phonetic Decision Trees

Our goals in building the phonetic decision tree code were to make it efficient for arbitrary context sizes (i.e. we avoided enumerating contexts), and also to make it general enough to support a wide range of approaches. The conventional approach is, in each HMM-state of each monophone, to have a decision tree that asks questions about, say, the left and right phones. In our framework, the decision-tree roots can be shared among the phones and among the states of the phones, and questions can be asked about any phone in the context window, and about the HMM state. Phonetic questions can be supplied based on linguistic knowledge, but in our recipes the questions are generated automatically based on a tree-clustering of the phones. Questions about things like phonetic stress (if marked in the dictionary) and word start/end information are supported via an extended phone set; in this case we share the decision-tree roots among the different versions of the same phone.

VI. Language Modeling

Since Kaldi uses an FST-based framework, it is possible, in principle, to use any language model that can be represented as an FST. We provide tools for converting LMs in the standard ARPA format to FSTs. In our recipes, we have used the IRSTLM toolkit [3] for purposes like LM pruning. For building LMs from raw text, users may use the IRSTLM toolkit, for which we provide installation help, or a more fully-featured toolkit such as SRILM [4].

VII. Creating Decoding Graphs

All our training and decoding algorithms use Weighted Finite State Transducers (WFSTs). In the conventional recipe [14], the input symbols on the decoding graph correspond to context-dependent states (in our toolkit, these symbols are numeric and we call them pdf-ids). However, because we allow different phones to share the same pdf-ids, we would have a number of problems with this approach, including not being able to determinize the FSTs, and not having sufficient information from the Viterbi path through an FST to work out the phone sequence or to train the transition probabilities. In order to fix these problems, we put on the input of the FSTs a slightly more fine-grained integer identifier that we call a "transition-id", that encodes the pdf-id, the phone it is a member of, and the arc (transition) within the topology specification for that phone. There is a one-to-one mapping between the "transition-ids" and the transition-probability parameters in the model: we decided make transitions as fine-grained as we could without increasing the size of the decoding graph.

Our decoding-graph construction process is based on the recipe described in [14]; however, there are a number of differences. One important one relates to the way we handle "weight-pushing", which is the operation that is supposed to ensure that the FST is stochastic. "Stochastic" means that the weights in the FST sum to one in the appropriate sense, for each state (like a properly normalized HMM). Weight pushing may fail or may lead to bad pruning behavior if the FST representing the grammar or language model (G) is not stochastic, e.g. for backoff language models. Our approach is to avoid weight-pushing altogether, but to ensure that each stage of graph creation "preserves stochasticity" in an appropriate sense. Informally, what this means is that the "non-sum-to-one-ness" (the failure to sum to one) will never get worse than what was originally present in G.

TABLE I
Basic triphone system on Resource Management: %WERs

	Test set				
	Feb'89	Oct'89	Feb'91	Sep'92	Avg
HTK	2.77	4.02	3.30	6.29	4.10
Kaldi	3.20	4.21	3.50	5.86	4.06

VIII. Decoders

We have several decoders, from simple to highly optimized; more will be added to handle things like on-the-fly language model rescoring and lattice generation. By "decoder" we mean a C++ class that implements the core decoding algorithm. The decoders do not require a particular type of acoustic model: they need an object satisfying a very simple interface with a function that provides some kind of acoustic model score for a particular (input-symbol and frame).

```
class DecodableInterface {
 public:
  virtual float LogLikelihood(int frame, int index) = 0;
  virtual bool IsLastFrame(int frame) = 0;
  virtual int NumIndices() = 0;
  virtual ~DecodableInterface() {}
};
```

Command-line decoding programs are all quite simple, do just one pass of decoding, and are all specialized for one decoder and one acoustic-model type. Multi-pass decoding is implemented at the script level.

IX. Experiments

We report experimental results on the Resource Management (RM) corpus and on Wall Street Journal. The results reported here correspond to version 1.0 of Kaldi; the scripts that correspond to these experiments may be found in `egs/rm/s1` and `egs/wsj/s1`.

A. *Comparison with previously published results*

Table I shows the results of a context-dependent triphone system with mixture-of-Gaussian densities; the HTK baseline numbers are taken from [15] and the systems use essentially the same algorithms. The features are MFCCs with per-speaker cepstral mean subtraction. The language model is the word-pair bigram language model supplied with the RM corpus. The WERs are essentially the same. Decoding time was about 0.13×RT, measured on an Intel Xeon CPU at 2.27GHz. The system identifier for the Kaldi results is tri3c.

Table II shows similar results for the Wall Street Journal system, this time without cepstral mean subtraction. The WSJ corpus comes with bigram and trigram language models. and we compare with published numbers using the bigram language model. The baseline results are reported in [16], which we refer to as "Bell Labs" (for the authors' affiliation), and a HTK system described in [17]. The HTK system was gender-dependent (a gender-independent baseline was not reported), so the HTK results are slightly better. Our decoding time was about 0.5×RT.

[3] Available from: http://hlt.fbk.eu/en/irstlm

[4] Available from: http://www.speech.sri.com/projects/srilm/

TABLE II
BASIC TRIPHONE SYSTEM, WSJ, 20K OPEN VOCABULARY, BIGRAM LM, SI-284 TRAIN: %WERS

	Test set	
	Nov'92	Nov'93
Bell	11.9	15.4
HTK (+GD)	11.1	14.5
KALDI	11.8	15.0

TABLE III
RESULTS ON RM AND ON WSJ, 20K OPEN VOCABULARY, BIGRAM LM, TRAINED ON HALF OF SI-84: %WERS

	RM (Avg)	WSJ Nov'92	WSJ Nov'93
Triphone	3.97	12.5	18.3
+ fMLLR	3.59	11.4	15.5
+ LVTLN	3.30	11.1	16.4
Splice-9 + LDA + MLLT	3.88	12.2	17.7
+ SAT (fMLLR)	2.70	9.6	13.7
+ SGMM + spk-vecs	2.45	10.0	13.4
+ fMLLR	2.31	9.8	12.9
+ ET	2.15	9.0	12.3

B. Other experiments

Here we report some more results on both the WSJ test sets (Nov'92 and Nov'93) using systems trained on just the SI-84 part of the training data, that demonstrate different features that are supported by Kaldi. We also report results on the RM task, averaged over 6 test sets: the 4 mentioned in table I together with Mar'87 and Oct'87. The best result for a conventional GMM system is achieved by a SAT system that splices 9 frames (4 on each side of the current frame) and uses LDA to project down to 40 dimensions, together with MLLT. We achieve better performance on average, with an SGMM system trained on the same features, with speaker vectors and fMLLR adaptation. The last line, with the best results, includes the "exponential transform" [12] in the features.

X. CONCLUSIONS

We described the design of Kaldi, a free and open-source speech recognition toolkit. The toolkit currently supports modeling of context-dependent phones of arbitrary context lengths, and all commonly used techniques that can be estimated using maximum likelihood. It also supports the recently proposed SGMMs. Development of Kaldi is continuing and we are working on using large language models in the FST framework, lattice generation and discriminative training.

ACKNOWLEDGMENTS

We would like to acknowledge participants and collaborators in the 2009 Johns Hopkins University Workshop, including Mohit Agarwal, Pinar Akyazi, Martin Karafiat, Feng Kai, Ariya Rastrow, Richard C. Rose and Samuel Thomas; Patrick Nguyen, for introducing the participants in that workshop and for help with WSJ recipes, and faculty and staff at JHU for their help during that workshop, including Sanjeev Khudanpur, Desirée Cleves, and the late Fred Jelinek.

We would like to acknowledge the support of Geoffrey Zweig and Alex Acero at Microsoft Research, and Dietrich Klakow at Saarland University. We are grateful to Jan (Honza) Černocký for helping us organize the workshop at the Brno University of Technology during August 2010 and 2011. Thanks to Tomas Kašpárek for system support and Renata Kohlová for administrative support.

We would like to thank Michael Riley, who visited us in Brno to deliver lectures on finite state transducers and helped us understand OpenFst; and Henrique (Rico) Malvar of Microsoft Research for allowing the use of his FFT code. We would like to acknowledge the help with coding and documentation from Sandeep Boda and Sandeep Reddy (sponsored by Go-Vivace Inc.) and Haihua Xu. We thank Pavel Matejka (and Phonexia s.r.o.) for allowing the use of feature processing code.

During the development of Kaldi, Arnab Ghoshal was supported by the European Community's Seventh Framework Programme under grant agreement no. 213850; the BUT researchers were supported by the Technology Agency of the Czech Republic under project No. TA01011328, and partially by Czech MPO project No. FR-TI1/034. The JHU 2009 workshop was supported by NSF Grant Number IIS-0833652, with supplemental funding from Google Research, DARPA's GALE program and the JHU Human Language Technology Center of Excellence.

REFERENCES

[1] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland, *The HTK Book (for version 3.4)*. Cambridge University Engineering Department, 2009.

[2] A. Lee, T. Kawahara, and K. Shikano, "Julius – an open source real-time large vocabulary recognition engine," in *EUROSPEECH*, 2001, pp. 1691–1694.

[3] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A flexible open source framework for speech recognition," Sun Microsystems Inc., Technical Report SML1 TR2004-0811, 2004.

[4] D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter, and H. Ney, "The RWTH Aachen University Open Source Speech Recognition System," in *INTERSPEECH*, 2009, pp. 2111–2114.

[5] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri, "OpenFst: a general and efficient weighted finite-state transducer library," in *Proc. CIAA*, 2007.

[6] R. Gopinath, "Maximum likelihood modeling with Gaussian distributions for classification," in *Proc. IEEE ICASSP*, vol. 2, 1998, pp. 661–664.

[7] M. J. F. Gales, "Semi-tied covariance matrices for hidden Markov models," *IEEE Trans. Speech and Audio Proc.*, vol. 7, no. 3, pp. 272–281, May 1999.

[8] C. J. Leggetter and P. C. Woodland, "Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models," *Computer Speech and Language*, vol. 9, no. 2, pp. 171–185, 1995.

[9] M. J. F. Gales, "Maximum likelihood linear transformations for HMM-based speech recognition," *Computer Speech and Language*, vol. 12, no. 2, pp. 75–98, April 1998.

[10] ——, "The generation and use of regression class trees for MLLR adaptation," Cambridge University Engineering Department, Technical Report CUED/F-INFENG/TR.263, August 1996.

[11] D. Y. Kim, S. Umesh, M. J. F. Gales, T. Hain, and P. C. Woodland, "Using VTLN for broadcast news transcription," in *Proc. ICSLP*, 2004, pp. 1953–1956.

[12] D. Povey, G. Zweig, and A. Acero, "The Exponential Transform as a generic substitute for VTLN," in *IEEE ASRU*, 2011.

[13] D. Povey, L. Burget *et al.*, "The subspace Gaussian mixture model—A structured model for speech recognition," *Computer Speech & Language*, vol. 25, no. 2, pp. 404–439, April 2011.

[14] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech and Language*, vol. 20, no. 1, pp. 69–88, 2002.

[15] D. Povey and P. C. Woodland, "Frame discrimination training for HMMs for large vocabulary speech recognition," in *Proc. IEEE ICASSP*, vol. 1, 1999, pp. 333–336.

[16] W. Reichl and W. Chou, "Robust decision tree state tying for continuous speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 5, pp. 555–566, September 2000.

[17] P. C. Woodland, J. J. Odell, V. Valtchev, and S. J. Young, "Large vocabulary continuous speech recognition using HTK," in *Proc. IEEE ICASSP*, vol. 2, 1994, pp. II/125–II/128.