Open Source KALDI Automated Speech Recognition (ASR) Engine

This documentation provides a general description on the structure of the KALDI ASR Engine, with features, design, the types of algorithms used and overview of the Engine for better understanding.

FEATURES

KALDI automatic speech recognition framework consist of the following features:

- Finite-state transducers (FST), compiled against the OpenFst toolkit using it as a library.
- Extensive linear algebra support, matrix library that wraps standard BLAS and LAPACK routines.
- Extensible design, algorithms are provided in the most generic form.
- Open license, Licensed under Apache v2.0.

ACOUSTIC MODELING

The following conventional models (i.e. diagonal GMMs) and Subspace Gaussian Mixture Models (SGMMs) were used. GMMs are the foundation before going into Deep Neural Network (DNN) model training. For DNN acoustic model training, we use Time Delayed Neural Network (TDNN) and unidirectional Long Short Term Memory (LSTM). These combinations of TDNN and LSTM results in low latency of acoustic modelling.

A. Diagonal and full covariance Gaussian Mixture Models (GMMs)
B. GMM-based acoustic model
C. HMM Topology
D. Speaker Adaptation that supports speaker normalization using a linear approximation to VTLN, similar to, or conventional feature-level VTLN, or a more generic approach for gender normalization, “exponential transform”.
E. Subspace Gaussian Mixture Models (SGMMs)
F. Time Delayed Neural Network (TDNN)
G. Unidirectional Long Short Term Memory (LSTM)

LANGUAGE MODEL (LM)

KALDI uses an FST-based framework. The SRILM toolkit were used for LM pruning, when building LMs from raw text, users may use the SRILM toolkit.

DECODERS

A C++ class that handles 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() {}
};
```

ONLINE DECODING

GMM-based online-decoding procedure were used for easy evaluate word error rates. The inner decoder object, LatticeFasterDecode takes the decoding graph (as a FST), and the decodable object. The OnlineFasterDecoder setup has the ability to work out which words are going to be “inevitably” decoded regardless of what audio data comes in future. This setup is useful for online-transcription context.

Neural network based online decoding with iVectors is used. It is based on the foundation of GMM-based decoding. To be exact, we use online decoding with nnet3 models, which supports forward recurrent models such as LSTM. The online decoder **online2-wav-nnet3-latgen-faster** is used to transcribe wav file to its corresponding text.

FEATURE EXTRACTION

The feature extraction support most commonly used approaches such as VTLN, cepstral mean and variance normalization, LDA, STC/MLLT, HLDA, and so on.

In **online-feature.h**, classes are available for various components of feature extraction. All components inheriting from class **OnlineFeatureInterface**. **OnlineFeatureInterface** is a base class for online feature extraction. The interface specifies how the object provides the features to the caller (**OnlineFeatureInterface::GetFrame()**) and how it says how many frames are ready (**OnlineFeatureInterface::NumFramesReady()**), but does not say how it obtains those features. That is up to the child class.

In **online-feature.h** we define classes **OnlineMfcc** and **OnlinePlp** which are the lowest-level features. They have a member function **OnlineMfccOrPlp::AcceptWaveform(),** which the user should call when data is captured. All the other online feature types in **online-feature.h** are "derived" features, so they take an object of **OnlineFeatureInterface** in their constructor and get their input features through a stored pointer to that object.

The only part of the online feature extraction code in **online-feature.h** that is non-trivial is the cepstral mean and variance normalization (CMVN) (and note that the fMLLR, or linear transform, estimation is not trivial but the complexity lies elsewhere). We describe the CMVN below.

CEPSTRAL MEAN AND VARIANCE NORMALIZATION IN ONLINE DECODING

In the Kaldi scripts, cepstral mean and variance normalization (CMVN) is performed on per-speaker basis. In an online-decoding context, it is not possible for "non-causal" speech.

Hence, "moving-window" cepstral mean normalization by accumulating the mean over a moving window of 6 seconds. The class for computing, **OnlineCmvnOptions**, also has extra configuration variables such as speaker-frames (default: 600ms) and global-frames (default: 200ms). To specifically state the duration on the use to prior information from the same speaker. This improves the estimation for the first few seconds of each utterance. The program **apply-cmvn-online** can apply normalization as part of a training pipeline for training on matched features.

ADAPTATION IN ONLINE DECODING

The most commonly adaptation method used for speech recognition is feature-space Maximum Likelihood Linear Regression (fMLLR) or Constrained MLLR (CMLLR).

The fMLLR consists of the following mathematical parameters:

- Affine (linear + offset) transform of the features,
- the number of parameters is d * (d+1), where d is the final feature dimension (typically 40).

The top-level logic for this at the decoder level is mostly implemented in class **SingleUtteranceGmmDecoder**. To incrementally estimate an increasing number of transform parameters as we decode more data.

The fMLLR perform estimation periodically and compute lattice posteriors. The **OnlineGmmDecodingAdaptationPolicyConfig** class determine when to re-estimate fMLLR. On first utterance, the default estimation period is after 2 seconds. The period increases geometrically at a ratio with constant of 1.5 (e.g. 2 seconds, 3 seconds, 4.5 seconds, etc). Any later utterances estimation starts after 5 seconds, 10 seconds, 20 seconds and so on. For all utterances we estimate it at the end of the utterance.

ONLINE DECODING WITH nnet3 MODELS

Nnet3 models were used in the current KALDI decoding model, for more information on the older model, nnet2. Please visit the link https://kaldi-asr.org/doc/online_decoding.html for more information. In Kaldi 5.1 and later, online **nnet3** decoding supports "forward" recurrent models such as LSTMs, but not bidirectional ones like BLSTMs. In addition, online **nnet3** decoding with recurrent models may not give optimal results unless you use "Kaldi-5.1-style" configuration, including the "decay-time" option and specifying –extra-left-context-initial 0.

KALDI Engine Overview

Two external libraries that are freely available: OpenFst for finite-state framework, the other is numerical algebra libraries. The standard algebra libraries used was “Basic Linear Algebra Subroutines” (BLAS) and “Linear Algebra PACKage” (LAPACK) [2] (refer to Fig 1).

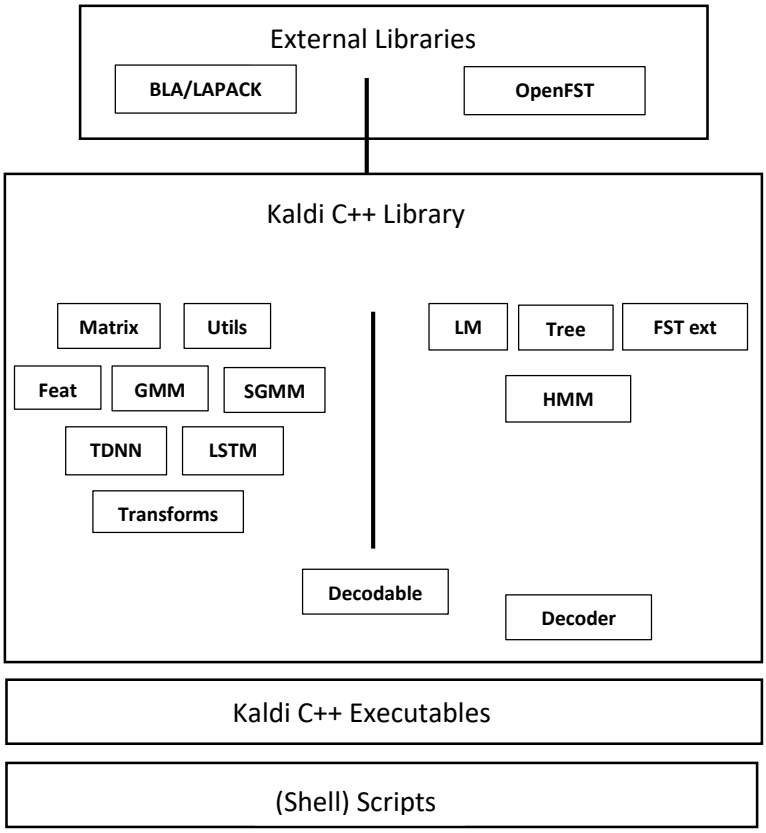

Fig 1. A simplified view of the different components of Kaldi.

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.