ExKaldi: A Python-based Extension Tool of Kaldi

Yu Wang, Chee Siang Leow, Hiromitsu Nishizaki
University of Yamanashi
Kofu, Yamanashi, Japan
{wangyu,cheesiang_leow, nisizaki}@alps-lab.org

Akio Kobayashi
Tsukuba University of Technology
Tsukuba, Ibaraki, Japan
a-kobayashi@a.tsukuba-tech.ac.jp

Takehito Utsuro
University of Tsukuba
Tsukuba, Ibaraki, Japan
utsuro@iit.tsukuba.ac.jp

***Abstract*—We present ExKaldi, an automatic speech recognition (ASR) toolkit, which is implemented based on the Kaldi toolkit and Python language. While similar Kaldi wrappers are available, a key feature of ExKaldi is an integrated strategy to build ASR systems, including processing feature and alignment, training an acoustic model, training, querying N-grams language model, decoding and scoring. Primarily, ExKaldi builds a bridge between Kaldi and deep learning frameworks to help users customize a hybrid hidden Markov model–deep neural network-based ASR system. We performed benchmark experiments on the TIMIT corpus and revealed that ExKaldi could build a system from scratch with Python and achieved reasonable recognition accuracy. The toolkit is open-source and released under the Apache license.**

***Index Terms*—automatic speech recognition, deep learning, Kaldi, Python,**

I. Introduction

In recent years, automatic speech recognition (ASR) technologies have achieved unprecedented progress. Kaldi [1], as one of the free and open-source toolkits for ASR, plays a remarkable role in various ASR tasks. It provides integrated, flexible libraries and tools written in C++ language. Besides, it also provides shell-scripts named "recipes," which allow the training of a variety of acoustic models.

Moreover, some open-source deep learning (DL) frameworks, such as TensorFlow [1] and PyTorch [2], have been developed. Python language has become the mainstream of these frameworks because of its usability and variable extension packages for machine learning. Owing to these tools, some excellent ASR systems, such as ESPNet [2] that adopted a deep neural network (DNN)-based model, have been developed and gained much attention.

At present, traditional ASR systems that are a combination of acoustic and language models are vital, especially those based on the hybrid of hidden Markov model (HMM) and DNN. Kaldi toolkit helps us to realize such a hybrid acoustic model. However, it is hard to train DNN-based acoustic models with various structures using Kaldi without specialized knowledge of Kaldi and C++. Therefore, a Kaldi-based ASR system with Python language is being demanded. Consequently, we have developed ExKaldi—-a Python-based extension tool of Kaldi.

Python-based wrappers of Kaldi have been released already; PyKaldi [3] and PyKaldi2 [4] are Python wrappers of Kaldi and open-source ASR toolkits. Unlike these two toolkits, ExKaldi does not require users to write the code with the Kaldi command format. It is friendlier to Python programmers than PyKaldi. Another related project is PyTorch-Kaldi [5]. However, PyTorch-Kaldi only focuses on designing and implementing the DNN acoustic model with PyTorch. Torchaudio [3], developed by the PyTorch team, has wrapped a part of Kaldi tools but does not support training Gaussian mixture model (GMM)-HMM model.

ExKaldi aims to help Python programmers build a Kaldi-based integral ASR system and flexibly adjust its details to enhance performance; it is available on GitHub [4].

II. ExKaldi Design

A. Data Structures in ExKaldi

We implement three basic data structures: *BytesArchive* and *ListTable*–designed to communicate with Kaldi–and *NumpyArchive*–contributing to DL. *BytesArchive* holds Kaldi binary archive-table, such as features which need to be processed frequently, and they can be obtained with both Kaldi and ExKaldi tools. Basically, ExKaldi interacts with Kaldi compiled C++ command-line interface (CLI) through the Python subprocess pipeline. Similar to other binary objects like the GMM-HMM model and lattice in ExKaldi, these binary archives will be sent to or read from standard I/O stream. Their formats can be quickly converted between Kaldi binary archives and Python NumPy [5] arrays. *NumpyArchive* also plays this role and further supports the training of DNN model with a DL framework. *ListTable* corresponds to Kaldi tables of text format. With these three base classes, we designed a group of subclasses to control data format and sequence strictly. They reduce much effort to check data format in Kaldi tasks and possible buffer costs when reading data from the stream. It enables Python programmers to build ASR systems with ease.

B. GMM-HMM Training

ExKaldi has wrapped relatively complete tools for training standard GMM-HMM based acoustic models, including training mono-phone GMM-HMM, decision tree, and context-phone GMM-HMM with the extracted acoustic feature, such as mel-frequency cepstrum coefficient (MFCC) and perceptual linear predictive (PLP) feature. It is a key feature of ExKaldi that differs from other Kaldi wrappers.

The standard feature optimization technologies, linear discriminant analysis (LDA), and maximum likelihood linear

[1]https://www.tensorflow.org/
[2]https://www.pytorch.org/
[3]https://pytorch.org/audio/
[4]https://github.com/wangyu09/exkaldi
[5]https://numpy.org/

978-1-7281-9802-6/20/$31.00 ©2020 IEEE

transform (MLLT) are also available. In addition, ExKaldi supports speaker adaptive training (SAT) by estimating the feature-space maximum likelihood linear regression (fMLLR) feature. Although all of these functions are implemented by the Kaldi backend, ExKaldi provides a flexible interface to train the GMM-HMM model that benefits from the excellent interactivity of Python. During each stage of the GMM-HMM training, users can easily tune the hyperparameters and optimize alignment and feature.

C. DL Support

ExKaldi can quickly switch Kaldi archive-table data to NumPy format, typically convert feature and alignment, and use them for training DNN-based acoustic model with a DL framework. Different categories of features and alignments can be grouped in various ways to support multiple tasks and generate an iterable dataset. Besides the popular probability distribution function (PDF) IDs and phone IDs alignment, user-defined labels like emotion and gender can also be paired with these data. ExKaldi will split them and take turns in loading them into the computer's memory parallelly. It helps to train a large-scale corpus.

D. Language Model

ExKaldi uses KenLM[6] and SRILM[7] as backends to train the N-grams language model. KenLM outperforms other similar toolkits, including SRILM, in memory management. In addition, ExKaldi supports the query and evaluation of language models. It helps users to build a better model, and thus, improves ASR accuracy.

In our experiment, we revealed that ExKaldi can well train and evaluate N-grams language models on both small-sized and large-sized text corpora.

E. Decoding and Scoring

ExKaldi provides an efficient way to manage lexicons. Various lexicons, such as **lexiconp** and **silence_phones**, are generated automatically and stored in memory. With the HMM and language model, ExKaldi can compile the weighted finite-state transducer (WFST) graph, and implement GMM decoding with the feature or DNN decoding with the output probability of DNN acoustic model. After the decoding, ExKaldi can deal with the generated lattice to optimize ASR results. Besides the decoding algorithm based on WFST, ExKaldi provides basic end-to-end decoding algorithms, such as beam search & language model, which is implemented in C++. Furthermore, ExKaldi can evaluate the ASR system by computing the frequently-used word error rate (WER) score or edit distance score.

F. Parallel Processes

Although we use *BytesArchive* to hold binary data in memory directly in general, *ArkIndexTable* is a subclass for holding the location index information and an another approach to describe binary archives. It allows ExKaldi to handle large-scale corpus and execute parallel processes by storing archives in files and using the index table to access them. ExKaldi provides an easy-to-use interface to call parallel processes just by receiving multiple resources or different parameters. In the next section, we will illustrate an example of a parallel-process.

[6]https://kheafield.com/code/kenlm

[7]http://www.speech.sri.com/projects/srilm/

III. Example Code

```python
# example.py
import exkaldi
from exkaldi import args

args.discribe("This is an example program to \
        train a dummy monophone GMM-HMM model.")
args.add("--parallel",abbr="-p",dtype=int,default=4,
         discription="The number of parallel processes.")

# Parse command-line options and take a backup for debugging.
args.parse()
args.save("conf/train_mono.args")

# Extract and process feature.
feat = exkaldi.compute_mfcc(target="train/wav.scp",rate=16000)
feat = feat.add_delta(order=2)
print(feat.dim)

# Prepare lexicons.
lexicons = exkaldi.decode.graph.lexicon_bank(
                    pronFile="dict/pronunciation.txt",
                    silWords={"sil":"sil"},
                    unkSymbol={"sil":"sil"},
                    optionalSilPhone="sil"
                   )

# Make the GMM-HMM topology.
exkaldi.hmm.make_topology(lexicons=lexicons,
                          outFile="dict/topo",
                          numNonsilStates=3,
                          numSilStates=5)

# Make Lexicon fst.
exkaldi.decode.graph.make_L(lexicons=lexicons,
                          outFile="dict/L.fst",
                          useDisambigLexicon=False)

# Initialize a monophone GMM-HMM model.
model = exkaldi.hmm.MonophoneHMM(lexicons=lexicons)
model.initialize(feat=feat,topoFile="dict/topo")
print(model.info)

# Prepare transcription for training.
transcription = exkaldi.load_transcription("train/text")

# when using parallel processes,
# we only need to split resources into N chunks.
if args.parallel > 1:
  # Split feature.
  feat = feat.sort(by="utt").subset(chunks=args.parallel)
  # Split transcription depending on utterance IDs of feature.
  temp = []
  for f in feat:
      temp.append(transcription.subset(keys=f.utts))
  transcription = temp
else:
  feat = feat.sort(by="utt")
  transcription = transcription.sort(by="utt")


# Run the training loop.
model.train(feat=feat,
            transcription=transcription,
            LFile="dict/L.fst",
            tempDir="exp/mono",
            numIters=40,
            maxIterInc=35,
            totgauss=1000,
            realignIter=[1,2,3,4,5,7,10,15,20,30]
            )
print(model.info)
```

Fig. 1. Train a monophone GMM-HMM model with ExKaldi.

Figure 1 shows an example of a mono-phone GMM-HMM training with ExKaldi. This script first parses the command-line options. *args* has a global scope and can be saved into

978-1-7281-9802-6/20/$31.00 ©2020 IEEE

TABLE I
PERPLEXITY OF VARIOUS LANGUAGE MODELS FOR TIMIT

	N-grams				
	N=2	N=3	N=4	N=5	N=6
Baseline IRSTLM	14.41	—	—	—	—
ExKaldi SRILM	14.42	13.05	13.67	14.3	14.53
ExKaldi KenLM	14.39	12.75	12.75	12.7	**12.25**

TABLE II
PER [%] OF VARIOUS ASR SYSTEMS FOR TIMIT

	Mono	Tri1	Tri2	Tri3
Baseline 2-Grams	32.54	26.17	23.63	21.54
ExKaldi 2-Grams	32.53	25.89	23.63	21.43
ExKaldi 6-Grams	29.83	24.07	22.40	**20.03**

or loaded from file for debugging. Then, it extracts MFCC feature data from *wav.scp* file with a sampling frequency of *16* kHz. Because we did not specify the output file, the returned object *feat* will be a *BytesFeature* object. We simply process the features further by adding 2-order deltas. Next, we prepare the necessary lexicons. The generated object, *lexicons* on line 20, is a *LexiconBank* object. It automatically produced about 20 lexicons according to the tasks. *LexiconBank* object will play a crucial role in ExKaldi. Here, we use it to make the GMM-HMM topology and Lexicon FST. After preparing the transcription for training, all preparations have been made. We can use the high-level application programming interface (API) *model.train()* to run the training loop. This function can execute parallel processes only when it receives multiple resources. So, in this example, we split the feature and the corresponding transcription into N parts if the argument *args.parallel* is greater than *1*. This step will save the trained GMM-HMM model, generated decision tree and final alignment in the output directory. We can employ them to decode or train a better model.

IV. AUTOMATIC SPEECH RECOGNITION EXPERIMENT

We evaluated the ExKaldi toolkit primarily on the TIMIT[8] corpus. We used the *train* dataset for training and *test* dataset for evaluation. The machine configuration of the CPU was Core i7 6950X 3.0GHz; memory was 128 GB; GPU was GeForce GTX1080Ti 11GB; and OS was Ubuntu 18.04.

Firstly, we evaluated the language models trained with three toolkits. The Kaldi baseline built a 2-grams model with the IRSTLM[9] toolkit, and we used ExKaldi to build higher-order N-grams models with both SRILM and KenLM backends. Table I shows the results of perplexity based on phoneme unit. Clearly, we can see that KenLM's 6-grams model is the best language model in our experiment, where the score is 12.25. Next, we trained the GMM-HMM models using the same recipe as the Kaldi baseline. We used the above best-trained language model to compile the decoding graph. Table II shows the phone error rates (PERs) [10] of the various ASR systems. The best PER is 20.03% after the SAT. Based on the experiment results, we built the DNN-HMM hybrid ASR systems. We used the fMLLR features extracted by our toolkit and built two sorts of neural network models: DNN model with dense layer only and long short-term memory (LSTM) model with TensorFlow. Using similar model structures and hyperparameters, we compared the results of our toolkit with those of the Kaldi baseline and the PyTorch-Kaldi toolkit. The Kaldi baseline DNN model is Karel ' s DNN model[11]. The version of the PyTorch-Kaldi toolkit is 1.0. Since the Pytorch-Kaldi toolkit does not have tools to align data, we used the Kaldi's alignments of *tri3* and *dnn4* to train its DNN and LSTM models, respectively. Table III lists the PERs of these ASR systems. Owing to some advanced DL techniques like batch normalization (BN) provided by the DL framework, we achieved a significant improvement. After the DNN training, we aligned the feature again and generate a better alignment. Furthermore, using chronological information, we obtained the best PER (PER=15.11%) with the LSTM model.

TABLE III
PER [%] OF TWO NEURAL NETWORK ARCHITECTURES FOR TIMIT

	DNN	LSTM
Kaldi Baseline	18.67	—
PyTorch-Kaldi	17.99	17.01
ExKaldi	16.23	**15.11**

[8]https://catalog.ldc.upenn.edu/LDC93S1

TABLE IV
PERPLEXITY OF VARIOUS LANGUAGE MODELS FOR CSJ

	3-grams	4-grams	5-grams
Baseline SRILM	67.89	—	—
ExKaldi SRILM	67.89	66.39	66.24
ExKaldi KenLM	67.51	66.02	**65.90**

In addition, in order to assess the effectiveness of supporting these language model toolkits on a large-sized text we used CSJ[12] corpus. We trained with *train* dataset and tested with *eval1* dataset. Table IV lists the perplexity scores based on word unit. It shows that ExKaldi can well support these toolkits.

V. CONCLUSIONS

In this paper, we proposed the ExKaldi toolkit–a Python-based wrapper of Kaldi. We revealed that ExKaldi provides a set of integrated tools to support GMM-HMM acoustic model, N-grams language model, DNN-HMM acoustic model, decoding, and scoring. As experimental results, we successfully built various ASR systems with the ExKaldi toolkit, and they exhibited good performances. We will improve ExKaldi to per-

[9]https://hlt-mt.fbk.eu/technologies/irstlm

[10]In our experiment, we used the Kaldi ' s **compute-wer** tool to calculate PER.

[11]Karel's DNN: One of TIMIT DNN architecture. https://kaldi-asr.org/doc/dnn1.html

[12]https://pj.ninjal.ac.jp/corpus_center/csj

978-1-7281-9802-6/20/$31.00 ©2020 IEEE

form online ASR in the next phase and support discriminative training in the future.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 20H05558. Besides, a part of this work was also supported by the Hoso Bunka Foundation.

REFERENCES

[1] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlíček, Y. Qian, P. Schwarz, J. Silovský, G. Stemmer and K. Vesel, "The Kaldi speech recognition toolkit," Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding, 2011.

[2] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba, Y. Unno, N. E. Y. Soplin, J. Heymann, M. Wiesner, N. Chen, A. Renduchintala and T. Ochiai, "ESPnet: End-to-End Speech Processing Toolkit," Proceedings of INTERSPEECH2018, 2018, pp.2207-2211.

[3] D. Can, V. Martinez, P. Papadopoulos and S. Narayanan, "Pykaldi: A Python Wrapper for Kaldi," Proceedings of ICASSP2018, 2018, pp. 5889-5893.

[4] L. Lu, X. Xiao, Z. Chen and Y. Gong, "Pykaldi2: Yet another speech toolkit based on Kaldi and Pytorch," reprint arXiv:1907.05955, 2019.

[5] M. Ravanelli, T. Parcollet and Y. Bengio, "The Pytorch-kaldi Speech Recognition Toolkit," Proceedings of ICASSP2019, 2019, pp. 6465–6469.

978-1-7281-9802-6/20/$31.00 ©2020 IEEE