IPSJ SIG Technical Report Vol.2019-SLP-130 No.5
2019/12/6

Development and Evaluation of Kaldi Extension Tools with Python

Yu Wang[1,a)] Hiromitsu Nishizaki[1,b)] Akio Kobayashi[2,c)] Takehito Utsuro[3,d)]

Abstract:
We have been developing extension tools of Kaldi, an automatic speech recognition toolkit, with Python language. A part of this toolkit works as wrapper of Kaldi, therefore, some operations, taking feature extraction and decoding with lattice as examples, are easily performed with Python code. In addition, our tools support training an acoustic model by using deep learning framework, such as Chainer. We evaluated our tools on TIMIT corpus and so on, and got a better ASR performance than other ASR systems in some ways.

Keywords: automatic speech recognition, deep learning, Kaldi, Python

1. Introduction

In recent decades, automatic speech recognition (ASR) [1] technologies have achieved unprecedented progress. Kaldi toolkit [2], as one of free and open-source ASR toolkits, is playing a crucial part in various ASR tasks. It provided a sets of integrated, flexible libraries and tools written in C++ language. With Kaldi, some state-of-the-art ASR systems [3][4][5] were developed. Recently, instead of those whose acoustic model consists of Hidden Markov Model and Gaussian Mixture Model (HMM-GMM), A lot of ASR systems started to introduce Deep Neural Network (DNN) into their acoustic model that are called HMM-DNN [6], and obtained better performances in some ways. Deep learning [7] showed its persuasive power in the progressive improvement of ASR technologies.

Concerning deep learning, the developments of various open-source frameworks such as TensorFlow[8], PyTorch[9], and Chainer [10], are of special importance. In addition, Python language has become mainstream of these frameworks, because of its usability and variable extension packages for machine learning. Thanks to these tools, some DNN-based ASR systems have sprung up like mushrooms and gain much attention. ESPnet [11], for instance, is an end-to-end speech recognition toolkit. It is different from traditional ASR technologies, and mainly focuses on speech-to-text and text-to-speech by using a DNN model directly. However, traditional ASR systems that are a combination of acoustic model and language model are still vital nowadays, especially those which adopted the hybrid of HMM and DNN. Therefore, such tools that support building DNN-based acoustic model even a whole ASR system with Python language, are being demanded.

PyKaldi [12] is a Python wrapper of the Kaldi toolkit and is also a free and open-source ASR toolkit. It supports users in implementing Kaldi-like tools with Python. Differing from our developing toolkit, which will be introduced later, PyKaldi does not designedly support deep learning frameworks to train a DNN-based acoustic model. Another Python project, Pytorch-Kaldi [13], is a toolkit that bridges the gap between the Kaldi toolkit and frameworks by Python language. It makes users able to process features and decode with the Kaldi toolkit, and build a DNN-based acoustic model with PyTorch, which is one of the popular deep learning frameworks. It provides serval ready-to-use DNN models and supports users to customize them. While the concept of our toolkit is similar to the PyTorch-Kaldi toolkit, we more aim to provide integrated and independent tools to help ASR users customize the ASR system in a flexible way, including processing feature, building DNN-based acoustic model, decoding and handling lattice, etc. In addition, we tend to adapt more deep learning frameworks. Our toolkit is named ”ExKaldi”.

The ExKaldi focuses on improving the user experience to train an acoustic model and build an ASR system as simple as possible. Furthermore, distinct from almost ASR toolkits, we allow ASR users to record their speech from a microphone and recognize it by users' customized ASR system.

We evaluated our toolkit on the TIMIT corpus. Serval acoustic models were trained with PyTorch and Chainer,

[1] Integrated Graduate School of Medicine,Engineering and Agricultural Sciences,University of Yamanashi, Japan
[2] Department of Industrial Information, Tsukuba University of Technology, Japan
[3] Graduate School of Systems and Informaation Engineering, University of Tsukuba, Japan
[a)] wangyu@alps-lab.org
[b)] hnishi@yamanashi.ac.jp
[c)] a-kobayashi@a.tsukuba-tech.ac.jp
[d)] utsuro@iit.tsukuba.ac.jp

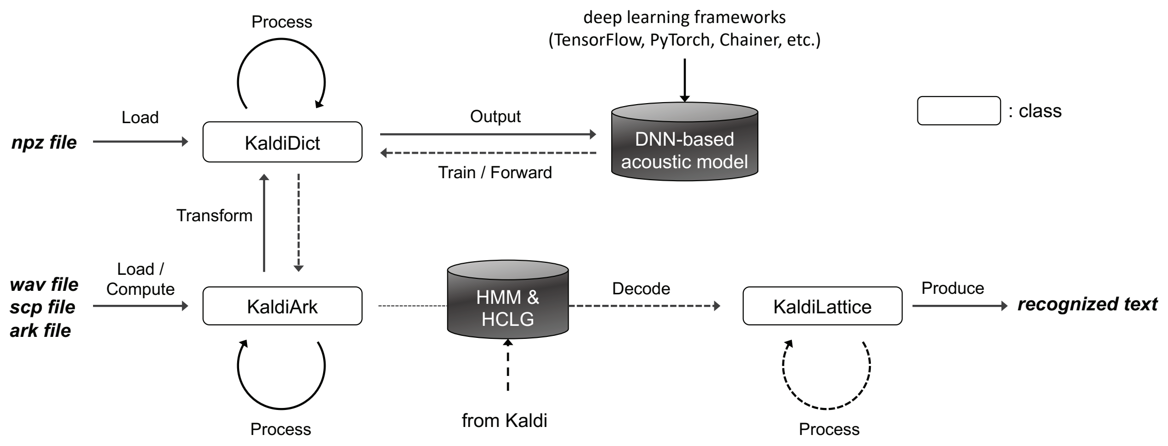

Fig. 1 An overview of the main usage of the ExKaldi toolkit.

major deep learning frameworks. As a result, the ExKaldi could achieve almost the same performances as the PyTorch-Kaldi results.

The remaining of this paper is organized as follows. In section 2, we will describe the main architecture of the ExKaldi toolkit. Then, section 3 will show an example code to bilud a simple ASR system from feature extracting to decoding and scoring. The section 4 will present the experimental results of servel DNN-based ASR systems built with our toolkit and compared their performances with other ASR systems. Finally, we will summarize the contribution of our research and propose the next plan in section 5. References will be listed in section 6.

2. Development of Kaldi Extension Tools

Two basic classes, *KaldiArk* and *KaldiDict*, are designed to hold features, alignments and posterior probability of acoustic model with the binary format and NumPy[*1] data respectively. Both *KaldiArk* object and *KaldDict* object can transform to each other directly. Based on this, we can finish training an acoustic model and building an HMM-DNN-based ASR system. With respect to decoding, we designed the third basic class, *KaldiLattice*, to carry lattice object in memory in order to further deal with it, such as scaling lattice and getting 1-best recognized result.

An overview of the primary usage of the ExKaldi toolkit is shown in Fig. 1. We introduce our ExKaldi toolkit in detail in the next sections.

2.1 Data Loading and Saving

We provided functions to acquire feature data, such as Mel-Frequency Cepstral Coefficients (MFCC) feature, from WAVE file(s) or Kaldi *wav-scp* file[*2]. Existed *ark* file[*3], and *feature-scp* file[*4] can also be loaded to a computer's memory easily. In addition, *KaldiDict* object can be load from *npz* (NumPy compressed) format file which has a specified format. Especially, to making labels, forced-alignment file(s) can be loaded and managed uniformly as *KaldiDict* object. *KaldiArk* object can be saved as *ark* file, and *KaldiDict* object can be saved as *npz* file.

2.2 KaldiArk and KaldiDict Class

The *KaldiArk* class is designed as an interface with the Kaldi toolkit. It can, therefore, hold the binary data in the same format as a Kaldi *ark* file. The *KaldiDict* class, which is a subclass of Python *dict* class, is the visual form of *KaldiArk*. Its keys are utterance names, and its values are acoustic features, forced-alignment data or probability of acoustic model presented with NumPy array format. *KaldiArk* and *KaldiDict* objects have some mutual attributes and methods, such as showing all utterances' names or splicing front-behind n frames. Because we designed the *KaldiDict* class to support deep learning, an *KaldiDict* object has several particular functions, for instance, normalizing all data to the standard normal distribution, generating iterative dataset quickly, etc. A part of our tools will return *KaldiArk* object or *KaldiDict* object.

2.3 Feature and Label Processing

We support further processing acoustic features, such as applying Cepstral Mean and Variance Normalization (CMVN) [14], adding n orders deltas, etc. For labels, both context-dependent targets (probability density function of each state of triphone) and context-independent targets (phone IDs) can be obtained to perform different or multiple tasks [15]. As mentioned earlier, we manage labels with the *KaldiDict* class by default. Feature data and labels, therefore, can be grouped to generate training or validation data quickly.

[*1] NumPy is the fundamental package for scientific computing in Python. available from ⟨https://docs.scipy.org/doc/numpy/index.html⟩
[*2] *wav-scp* file: One of Kaldi script files. It lists the name and path of WAVE files.
[*3] *ark* file: One of Kaldi archive files. It saved the name and feature parameters with binary format
[*4] *feature-scp*: One of Kaldi script files. It lists the name and path of feature parameter files.

2.4 Deep Learning Support

Besides mentioned above, we provided some basic tools to try to improve deep learning performance. For instance, when users want to train a sequence acoustic model, such as Long Short-Term Memory (LSTM) [16] model and Gated Recurrent Unit (GRU) [17] model, they are able to truncat long sequences in order to update model parameters more easily at the beginning of training loops, and pad batch sequences by randomly choosing start position to improve robustness of model. Besides, the *DataIterator* class and the *Supporter* class are designed, and the former can accept Kaldi *scp* file and split it into n chunks, then manage and take turns in loading them to computer's memory parallelly, which makes it reliable to train a large-scale corpus even on a general computer. The *Supporter* object is used to observe and grasp the changes of expected values, such as training loss, during training a DNN model. This information will be used efficiently to implement some automatic operations, such as saving a model depending on training performance or adjusting the learning rate when updating the DNN's parameters.

2.5 Decoding and Scoring

We accept log-like outputs of acoustic model and decode it with lattice by cooperating with existed HMM and Weighted Finite-State Transducer (WFST) [18]. This operation will generate a *KaldiLattice* object. *KaldiLattice* provides some tools to further deal with lattice, such as scaling, adding insertion penalty and getting n best words or phone sequences of these words with/without likelihood. These are useful to help users fine-tune their ASR systems. *KaldiLattice* object can be saved as and loaded from a lattice file with Kaldi format.

In the end, we provided functions to compute the frequently-used Word Error Rate (WER) score or Edit Distance score between the decoding result and reference text.

3. Example Code

Fig. 2 is an example to bulid a simple ASR system. The script first extracts MFCC feature data from *wav-scp* file with a sampling frequency of *16* kHz. It outputs a *KaldiArk* object *feat_1*, which is next normalized with CMVN state. CMVN state can be calculated with our toolkit. Then append two orders change information between adjacent feature frames by *add_delta*, and obtain *feat_3*, which is a *KaldiArk* object whose *splice* method is used to splice front-behind *5* frames on itself. So far, feature data have been processed simply. By using *array* method, a *KaldiDict* object, *feat_5*, is obtained. It will be used to forward the neural network. After forwarding, a new *KaldiDict* object, *amp_1*, is produced and transformed back to *KaldiArk* object with its *ark* method. Then use *decode_lattice* function to decode it with HMM and HCLG graph. After this step, a *KaldiLattice* object, *lat*, is produced. In the end, we select 1-best result by calling its *get_1best* method and compute WER score with *wer* function.

```python
# example.py
import exkaldi as E

# Compute MFCC feature from the wav scp file
# "feat_1" is a KaldiArk object.
feat_1 = E.compute_mfcc('test_wav.scp',
                        rate=16000)

# Apply CMVN to MFCC feature.
# "feat_2" is a KaldiArk object.
cmvnState = 'test_cmvn.ark'
uttSpk = 'test_uttSpk'
feat_2 = E.use_cmvn(feat_1, cmvnState,uttSpk)

# Add 2 orders deltas and splice front-behind
# 5 frames.
# "feat_3" and "feat_4" are KaldiArk objects.
feat_3 = E.add_delta(feat_2, order=2)
feat_4 = feat_3.splice(5)

# Transform binary data to numpy data.
# "feat_5" is a KaldiDict object.
feat_5 = feat_4.array

# Forward the neural network acoustic model
# and get log-like NumPy array data.
# "amp_1" is a KaldiDict object.
amp_1 = E.KaldiDict()
for utt in feat_5.utts:
  amp_1[utt] = acoustic_mode(feat_5[utt])

# Transform NumPy array back to binary data.
# "amp_2" is a KaldiArk object.
amp_2 = amp_1.ark

# Decode and generate a lattice.
# "lat" is a KaldiLattice object.
# HMM and HCLG files are generated
# with the original Kaldi tools.
hmm = 'test_graph/final.mdl'
hclg= 'test_graph/HCLG.fst'
wordSymbol = 'test_graph/words.txt'
lat = E.decode_lattice(amp_2, hmm, hclg,
                       wordSymbol, acwt=1)

# Get 1-best result from the lattice and
# compute the WER score.
outs = lat.get_1best(acwt=0.2)
score = E.wer(hyp=outs, ref='reference.txt')
print(score['WER'])
```

Fig. 2 An example code for building a symple ASR system using the ExKaldi toolkit.

4. ASR Experiment

4.1 Experimental Setup

We evaluated the ExKaldi toolkit on the TIMIT corpus[19]. We built serval HMM-DNN hybrid ASR systems by using our toolkit. We used Maximum Likelihood Linear Regression fratures (fMLLR) which have been already extracted from the training data set and the core test data set by running the Kaldi TIMIT recipe up to *tri3*[*5]. Training data consists of 3,696 utterances of 3.14 hours from 462 speakers. It was used to train the DNN-based acous-

[*5] *tri3*: One of training steps of Kaldi TIMIT recipe, which applied Linear Discriminant Analysis (LDA) and Maximum Likelihood Linear Transformation (MLLT) and Speaker Adaptive Training (SAT).

Table 1 WER [%] for the test data set of the TIMIT with various architectures.

	DNN	LSTM	GRU
Kaldi beseline	18.67	—	—
PyTorch Kaldi	18.16	17.01	15.63
ExKaldi	16.94	16.44	15.61

tic model. Test data includes 192 utterances of 0.16 hours from 24 speakers. It was used to forward the model and decode with a lattice. And finally, we computed the WER score.

We evaluated three sorts of neural network models: DNN model (with dense layer only), LSTM model and GRU model. Both the DNN and LSTM models were programed with Chainer, and GRU is a Pytorch model. We would compare our architectures with the Kaldi TIMIT baseline and Pytorch-Kaldi's TIMIT architectures. The baseline of Kaldi DNN model is Karel ' s DNN model[*6]. With respected to the GRU model, we used the existed architecture which was developed in the PyTorch-Kaldi project. The version of the PyTorch-Kaldi toolkit is 0.1. All of our architectures and Pytorch-Kaldi's architectures are carried out with multiple tasks which used phone IDs as condition to restrict that neural network's outputs. Our machine had such configuration: CPU was Core i7 6950X 3.0GHz; Memory was 128 GB; GPU was GeForce GTX1080Ti 12GB; OS was Ubuntu 18.04.

4.2 Experimental Result

Table 1 shows the best performance we obtained of WER score on the TIMIT test data set. In terms of DNN, we did some fine operations, for example splicing more front-behind frames than both Kaldi and Pytorch-Kaldi model, and initializing network parameters by standard normal distribution, and manually reducing the learning rate. Thanks to these, we got a good performance (WER=16.94%). As a result of making use of speech sequence information, LSTM and GRU model did better jobs. We mainly implemented fewer nodes without Dropout[20] as well as using a large acoustic model weight to decode when training the LSTM model. Together with other settings, we have achieved a result with WER of 16.44%. Finally, we re-implemented the GRU model developed in Pytorch-Kaldi project with our toolkit and configured almost the same value of hyperparameters. It obtained a similar WER (WER=15.61%) score with the result of Pytorch-Kaldi.

5. Discussion

In this paper, we proposed the ExKaldi automatic speech recognition toolkit, which is based on the Kaldi toolkit. The ExKaldi provided a set of functions to support training a DNN-based acoustic model with Deep Learning frameworks and further customizing an ASR system easily with Python language. As experimental results, we built serval HMM-DNN systems with our toolkit, and they showed a good performance.

The ExKaldi tookit is released on

`https://github.com/wangyu09/exkaldi`

under the Apache License v2.0. We hope to improve ExKaldi toolkit to support computing fMLLR feature and making n-grams language model and HCLG graph in the next phase, then being able to implement more lattice operations and further support online recognition in the further. The ExKaldi toolkit is expected to contribute to more developers of automatic speech recognition technologies.

Acknowledgments
This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research, Grant Numbers 17H01977 and 19H01125.

[*6] Karel's DNN: One of TIMIT DNN architecture. available from ⟨https://kaldi-asr.org/doc/dnn1.html⟩.

References

[1] D. Yu and L. Deng: *Automatic Speech Recognition – A Deep Learning Approach*, Springer, 2015.

[2] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, " The Kaldi Speech Recognition Toolkit, " in *Proc. of ASRU 2011*, 2011.

[3] J. Guglani and A. Mishra, "Continuous punjabi speech recognition model based on kaldi asr toolkit, " in *Proc. of ICASSP 2018*, 2018, pp. 211–216.

[4] C. Piero, "A kaldi-dnn-based asr system for italian, " in *Proc. of IJCNN 2015*, 2015, pp.1–5.

[5] C. Luscher, E. Beck, K. Irie, M. Kitza, W. Michel, A. Zeyer, R. Schluter, and H. Ney, " RWTH ASR systems for librispeech: Hybrid vs attention, " in *Proc. of INTERSPEECH 2019*, 2019, pp.231–235.

[6] A. Mohamed, G. Dahl, and G. Hinton, " Acoustic modeling using deep belief networks, " in *IEEE Trans. on ASLP*, vol. 20, no. 1, 2010, pp. 14–22.

[7] I. Goodfellow, Y. Bengio, and A. Courville: *Deep Learning*, MIT Press, 2016, available from ⟨http://www.deeplearningbook.org⟩.

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.s Corrado, A. Davis, J. Dean, M. Devin, S.Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, and X. Zheng, " TensorFlow : Large-scale machine learning on learning on heterogeneous distributed systems, " 2015.

[9] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, " Automatic differentiation in Pytorch, " in *Proc. of NIPS 2017*, 2017, pp.1–4.

[10] S. Tokui, R. Okuta, T. Akiba, Y. Niitani, T. Ogawa,S. Saito, S. Suzuki, K. Uenishi, B. Vogel, andH. Yamazaki-Vincen, " Chainer: A deep learning framework for accelerating the research cycle, " in *Proc.of 25th ACM SIGKDD*, 2019, pp.2002–2011.

[11] S. Watanabe, T. Hori, S. Karita, T. Hayashi, J. Nishitoba,Y. Unno, N. Soplin, J. Heymann, M. Wiesner, N. Chen,A. Renduchintala, and T. Ochiai, "ESPnet: End-to-end speech processing toolkit, " in *Proc. of INTERSPEECH2018*, 2018, pp. 2207–2211.

[12] D. Can, V. R. Martinez, P. Papadopoulos, and S. S.Narayanan, " Pykaldi: A python wrapper for kaldi, " in *Proc. of ICASSP 2018*, 2018, pp. 5889–5893.

[13] M. Ravanelli, T. Parcollet, and Y. Bengio, " The PyTorch-Kaldi speech recognition toolkit, " in *Proc. of ICASSP 2019*, 2019, pp. 6465–6469.

[14] O. Strand and A. Egeberg, "Cepstral mean and variancenormalization in the model domain, " in *Proceedings of Research Workshop on Robustness Issues in Conversa-tional Interaction*, 2014, pp. 1–4.

[15] P. Bell, P. Swietojanski, and S. Renals, " Multitask learning of context-dependent targets in deep neural net-work acoustic models, " in *IEEE/ACM Trans. on ASLP* ,vol. 25, 2017, pp.238–247.

[16] H. Sak, A. Senior, and F. Beaufays, " Long short-termmemory recurrent neural network architectures for largescale acoustic modeling, " in *Proc. of INTER-*

SPEECH2014, 2014, pp.338–342.

[17] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Em-pirical evaluation of gated recurrent neural networks onsequence modeling,” in *Proc. of NIPS 2014 Workshopon Deep Learning*, 2014.

[18] T. Hori and A. Nakamura: *Speech Recognition Algorithms Using Weighted Finite-State Transducers*, Mor-gan & Claypool Publishers, 2013.

[19] J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett,N. Dahlgren, and V. Zue: *TIMIT Acoustic-Phonetic Continuous Speech Corpus* , Linguistic Data Con-sortium, 1993, available from ⟨https://catalog.ldc.upenn.edu/LDC93S1⟩.

[20] D. Kingma, T. Salimans, and M. Welling, “Variational dropout and the local reparameterization trick,” in *Proc.of NIPS 2016*, 2016, pp. 2575–2583