THE PYTORCH-KALDI SPEECH RECOGNITION TOOLKIT

Mirco Ravanelli[1], *Titouan Parcollet*[2], *Yoshua Bengio*[1*]

[1] Mila, Université de Montréal , *CIFAR Fellow
[2] LIA, Université d'Avignon

ABSTRACT

The availability of open-source software is playing a remarkable role in the popularization of speech recognition and deep learning. Kaldi, for instance, is nowadays an established framework used to develop state-of-the-art speech recognizers. PyTorch is used to build neural networks with the Python language and has recently spawn tremendous interest within the machine learning community thanks to its simplicity and flexibility.

The PyTorch-Kaldi project aims to bridge the gap between these popular toolkits, trying to inherit the efficiency of Kaldi and the flexibility of PyTorch. PyTorch-Kaldi is not only a simple interface between these software, but it embeds several useful features for developing modern speech recognizers. For instance, the code is specifically designed to naturally plug-in user-defined acoustic models. As an alternative, users can exploit several pre-implemented neural networks that can be customized using intuitive configuration files. PyTorch-Kaldi supports multiple feature and label streams as well as combinations of neural networks, enabling the use of complex neural architectures. The toolkit is publicly-released along with a rich documentation and is designed to properly work locally or on HPC clusters.

Experiments, that are conducted on several datasets and tasks, show that PyTorch-Kaldi can effectively be used to develop modern state-of-the-art speech recognizers.

Index Terms: speech recognition, deep learning, Kaldi, PyTorch.

1. INTRODUCTION

Over the last years, we witnessed a progressive improvement and maturation of Automatic Speech Recognition (ASR) technologies [1, 2], that have reached unprecedented performance levels and are nowadays used by millions of users worldwide.

A key role in this technological breakthrough is being played by deep learning [3], that contributed to overcoming previous speech recognizers based on Gaussian Mixture Models (GMMs). Beyond deep learning, other factors have played a role in the progress of the field. A number of speech-related projects such as AMI [4], DICIT [5], DIRHA [6] and speech recognition challenges such as CHiME [7], Babel, and Aspire, have remarkably fostered the progress in ASR. The public distribution of large datasets such as Librispeech [8] has also played an important role to establish common evaluation frameworks and tasks.

Among the others factors, the development of open-source software such as HTK [9], Julius [10], CMU-Sphinx, RWTH-ASR [11], LIA-ASR [12] and, more recently, the Kaldi toolkit [13] have further helped popularize ASR, making both research and development of novel ASR applications significantly easier.

Kaldi currently represents the most popular ASR toolkit. It relies on finite-state transducers (FSTs) [14] and provides a set of C++ libraries for efficiently implementing state-of-the-art speech recognition systems. Moreover, the toolkit includes a large set of recipes that cover all the most popular speech corpora. In parallel to the development of this ASR-specific software, several general-purpose deep learning frameworks, such as Theano [15], TensorFlow [16], and CNTK [17], have gained popularity in the machine learning community. These toolkits offer a huge flexibility in the neural network design and can be used for a variety of deep learning applications.

PyTorch [18] is an emerging python package that implements efficient GPU-based tensor computations and facilitates the design of neural architectures, thanks to proper routines for automatic gradient computation. An interesting feature of PyTorch lies in its modern and flexible design, that naturally supports dynamic neural networks. In fact, the computational graph is dynamically constructed on-the-fly at running time rather than being statically compiled.

The PyTorch-Kaldi project aims to bridge the gap between Kaldi and PyTorch[1]. Our toolkit implements acoustic models in PyTorch, while feature extraction, label/alignment computation, and decoding are performed with the Kaldi toolkit, making it a perfect fit to develop state-of-the-art DNN-HMM speech recognizers. PyTorch-Kaldi natively supports several DNNs, CNNs, and RNNs models. Combinations between deep learning models, acoustic features, and labels are also supported, enabling the use of complex neural architectures. For instance, users can employ a cascade between CNNs, LSTMs, and DNNs, or run in parallel several models that share some hidden layers. Users can also explore different acoustic features, context duration, neuron activations (e.g., ReLU, leaky ReLU), normalizations (e.g., batch [19] and layer normalization [20]), cost functions, regularization strategies (e.g, L2, dropout [21]), optimization algorithms (e.g., SGD, Adam [22], RMSPROP), and many other hyper-parameters of an ASR system through simple edits of a configuration file.

The toolkit is designed to make the integration of user-defined acoustic models as simple as possible. In practice, users can embed their deep learning model and conduct ASR experiments even without being fully familiar with the complex speech recognition pipeline. The toolkit can perform computations on both local machines and HPC cluster, and supports multi-gpu training, recovery strategy, and automatic data chunking.

The experiments, conducted on several datasets and tasks, have shown that PyTorch-Kaldi makes it possible to easily develop competitive state-of-the-art speech recognition systems.

2. THE PYTORCH-KALDI PROJECT

An overview of the architecture adopted in PyTorch-Kaldi is reported in Fig. 1. The main script *run_exp.py* is written in *python*

[1] The code is available on GitHub (`github.com/mravanelli/PyTorch-kaldi/`).

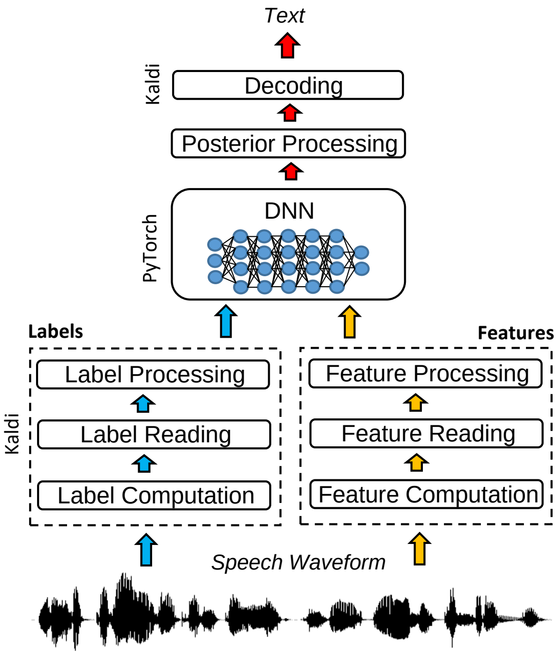

Fig. 1: An overview of the PyTorch-Kaldi architecture.

and manages all the phases involved in an ASR system, including feature and label extraction, training, validation, decoding, and scoring. The toolkit is detailed in the following sub-sections.

2.1. Configuration file

The main script takes as input a configuration file in INI format[2], that is composed of several sections. The section $[Exp]$ specifies some high-level information such as the folder used for the experiment, the number of training epochs, the random seed. It also allows users to specify whether the experiments have to be conducted on a CPU, GPU, or on multiple GPUs. The configuration file continues with the $[dataset*]$ sections, that specify information on features and labels, including the paths where they are stored, the characteristics of the context window [23], and the number of chunks in which the speech dataset must be split. The neural models are described in the $[architecture*]$ sections, while the $[model]$ section defines how these neural networks are combined. The latter section exploits a simple meta-language that is automatically interpreted by the $run\_exp.py$ script. Finally, the configuration file defines the decoding parameters in the $[decoding]$ section.

2.2. Features

The feature extraction is performed with Kaldi, that natively provides c++ libraries (e.g., *compute-mfcc-feats*, *compute-fbank-feats*, *compute-plp-feats*) to efficiently extract the most popular speech recognition features. The computed coefficients are stored in binary archives (with extension *.ark*) and are later imported into the python environment using the *kaldi-io* utilities inherited from the kaldi-io-for-python project[3]. The features are then processed by the function *load-chunk*, that performs context window composition, shuffling, as well as mean and variance normalization. As outlined before, PyTorch-Kaldi can manage multiple feature streams. For instance, users can define models that exploit combinations of MFCCs, FBANKs, PLP, and fMLLR [24] coefficients.

2.3. Labels

The main labels used for training the acoustic model derive from a forced alignment procedure between the speech features and the sequence of context-dependent phone states computed by Kaldi with a phonetic decision tree. To enable multi-task learning, PyTorch-kaldi supports multiple labels. For instance, it is possible to jointly load both context-dependent and context-independent targets and use the latter ones to perform monophone regularization [25, 26]. It is also possible to employ models based on an ecosystem of neural networks performing different tasks, as done in the context of joint training between speech enhancement and speech recognition [27–29] or in the context of the recently-proposed cooperative networks of deep neural networks [30].

2.4. Chunk and Mini-batch Composition

PyTorch-Kaldi automatically splits the full dataset into a number of chunks, which are composed of labels and features randomly sampled from the full corpus. Each chunk is then stored into the GPU or CPU memory and processed by the neural training algorithm $run\_nn.py$. The toolkit dynamically composes different chunks at each epoch. A set of mini-batches are then derived from them. Minibatches are composed of few training examples that are used for gradient computation and parameter optimization.

The way mini-batches are gathered strongly depends on the typology of the neural network. For feed-forward models, the minibatches are composed of randomly shuffled features and labels sampled from the chunk. For recurrent networks, the minibatches must be composed of full sentences. Different sentences, however, are likely to have different duration, making zero-padding necessary to form mini-batches of the same size. PyTorch-Kaldi sorts the speech sequences in ascending order according to their lengths (i.e., short sentences are processed first). This approach minimizes the need of zero-paddings and turned out to be helpful to avoid possible biases on batch normalization statistics. Moreover, it has been shown useful to slightly boost the performance and to improve the numerical stability of gradients.

2.5. DNN acoustic modeling

Each minibatch is processed by a neural network implemented with PyTorch, that takes as input the features and as outputs a set of posterior probabilities over the context-dependent phone states. The code is designed to easily plug-in customized models. As reported in the pseudo-code reported in Fig. 2, the new model can be simply defined by adding a new class into the *neural_nets.py*. The class must be composed of an initialization method, that specifies the parameters with their initialization, and a forward method that defines the computations to perform.

As an alternative, a number of pre-defined state-of-the-art neural models are natively implemented within the toolkit. The current version supports standard MLPs, CNNs, RNNs, LSTM, and GRU models. Moreover, it supports some advanced recurrent architectures, such as the recently-proposed Light GRU [31] and twin-regularized RNNs [32]. The SincNet model [33, 34] is also implemented to perform speech recognition from raw waveform directly. The hyperparameters of the model (such as learning rate, number of neurons,

[2] The configuration file is fully described in the project documentation.

[3] `github.com/vesis84/kaldi-io-for-python`

Fig. 2: Adding a user model into PyTorch-Kaldi

```
class my_NN(nn.Module):
    def __init__(self, options):
        super(my_NN, self).__init__()
        # Definition of Model Parameters
        # Parameter Initialization

    def forward(self, minibatch):
        # Definition of Model Computations
        return [output_prob]
```

number of layers, dropout factor, etc.) can be tuned using a utility that implements the random search algorithm [35].

2.6. Decoding and Scoring

The acoustic posterior probabilities generated by the neural network are normalized by their prior before feeding a HMM-based decoder. The decoder merges the acoustic scores with the language probabilities derived by an n-gram language model and tries to retrieve the sequence of words uttered in the speech signal using a beam-search algorithm. The final Word-Error-Rate (WER) score is computed with the NIST SCTK scoring toolkit.

3. EXPERIMENTAL SETUP

In the following sub-sections, the corpora, and the DNN setting adopted for the experimental activity are described.

3.1. Corpora and Tasks

The first set of experiments was performed with the TIMIT corpus, considering the standard phoneme recognition task (aligned with the Kaldi s5 recipe [13]).

To validate our model in a more challenging scenario, experiments were also conducted in distant-talking conditions with the DIRHA-English dataset[4] [36,37]. Training was based on the original WSJ-5k corpus (consisting of $7,138$ sentences uttered by 83 speakers) that was contaminated with a set of impulse responses measured in a domestic environment [37]. The test phase was carried out with the real part of the dataset, consisting of 409 WSJ sentences uttered in the aforementioned environment by six native American speakers.

Additional experiments were conducted with the CHiME 4 dataset [7], that is based on speech data recorded in four noisy environments (on a bus, cafe, pedestrian area, and street junction). The training set is composed of $43,690$ noisy WSJ sentences recorded by five microphones (arranged on a tablet) and uttered by a total of 87 speakers. The test set *ET-real* considered in this work is based on $1,320$ real sentences uttered by four speakers, while the subset *DT-real* has been used for hyperparameter tuning. The CHiME experiments were based on the single channel setting [7].

Finally, experiments were performed with the LibriSpeech [8] dataset. We used the training subset composed of 100 hours and the *dev-clean* set for the hyperparameter search. Test results are reported on the *test-clean* part using the *fglarge* decoding graph inherited from the Kaldi s5 recipe.

Table 1: PER(%) obtained for the test set of TIMIT with various neural architectures.

	MFCC	FBANK	fMLLR
MLP	18.2	18.7	16.7
RNN	17.7	17.2	15.9
LSTM	15.1	14.3	14.5
GRU	16.0	15.2	14.9
Li-GRU	15.3	14.6	**14.2**

3.2. DNN setting

The experiments consider different acoustic features, i.e., 39 MFCCs (13 static+Δ+$\Delta\Delta$), 40 log-mel filter-bank features (FBANKS), as well as 40 fMLLR features [24] (extracted as reported in the s5 recipe of Kaldi), that were computed using windows of 25 ms with an overlap of 10 ms.

The feed-forward models were initialized according to the *Glorot*'s scheme [38], while recurrent weights were initialized with orthogonal matrices [39]. Recurrent dropout was used as a regularization technique [40]. Batch normalization was adopted for feed-forward connections only, as proposed in [41, 42]. The optimization was done using the RMSprop algorithm running for 24 epochs. The performance on the development set was monitored after each epoch and the learning rate was halved when the relative performance improvement went below 0.1%. The main hyperparameters of the model (i.e., learning rate, number of hidden layers, hidden neurons per layer, dropout factor, as well as the twin regularization term λ) were tuned on the development datasets.

4. BASELINES

In this section, we discuss the baselines obtained with TIMIT, DIRHA, CHiME, and LibriSpeech datasets. As a showcase to illustrate the main functionalities of the PyTorch-Kaldi toolkit, we first report the experimental validation conducted on TIMIT.

Table 1 shows the performance obtained with several feed-forward and recurrent models using different features. To ensure a more accurate comparison between the architectures, five experiments varying the initialization seeds were conducted for each model and feature. The table thus reports the average *phone error rates* (PER)[5]. Results show that, as expected, fMLLR features outperform MFCCs and FBANKs coefficients, thanks to the speaker adaptation process. Recurrent models significantly outperform the standard MLP one, especially when using LSTM, GRU, and Li-GRU architecture, that effectively address gradient vanishing through multiplicative gates. The best result (PER=14.2%) is obtained with the Li-GRU model [31], that is based on a single gate and thus saves 33% of the computations over a standard GRU.

Table 2 details the impact of some popular techniques implemented in PyTorch-Kaldi for improving the ASR performance. The first row (Baseline) reports the performance achieved with a basic recurrent model, where powerful techniques such as dropout and batch normalization are not adopted. The second row highlights the performance gain that is achieved when progressively increasing the sequence length during training. In this case, we started the training by truncating the speech sentence at 100 steps (i.e, approximately 1 second of speech) and we progressively double the maximum sequence duration at every epoch. This simple strategy generally improves the

[4] This dataset is distributed by the Linguistic Data Consortium (LDC).

[5] Standard deviations range between 0.15 and 0.2 for all the experiments.

Table 2: PER(%) obtained on TIMIT when progressively applying some techniques implemented within PyTorch-Kaldi.

	RNN	LSTM	GRU	Li-GRU
Baseline	16.5	16.0	16.6	16.3
+ Incr. Seq. length	16.6	15.3	16.1	15.4
+ Recurrent Dropout	16.4	15.1	15.4	14.5
+ Batch Normalization	16.0	14.8	15.3	14.4
+ Monophone Reg.	15.9	14.5	14.9	**14.2**

Table 3: PER(%) obtained by combining multiple neural networks and acoustic features.

Architecture	Features	PER (%)
Li-GRU	fMLLR	14.2
MLP+Li-GRU+MLP	MFCC+FBANK+fMLLR	**13.8**

Table 4: PER(%) obtained with standard convolutional and with the SincNet architectures.

Model	Features	PER (%)
CNN	FBANK	18.3
CNN	Raw waveform	18.3
SincNet	Raw waveform	**18.1**

system performance since it encourages the model to first focus on short-term dependencies and learn longer-term ones only at a later stage. The third row shows the improvement achieved when adding recurrent dropout. Similarly to [40,42], we applied the same dropout mask for all the time steps to avoid gradient vanishing problems. The fourth line, instead, shows the benefits derived from batch normalization [19]. Finally, the last line shows the performance achieved when also applying monophone regularization [26]. In this case, we employ a multi-task learning strategy by means of two softmax classifiers: the first one estimates context-dependent states, while the second one predicts monophone targets. As observed in [26], our results confirm that this technique can successfully be used as an effective regularizer.

The experiments discussed so far are based on single neural models. In Table 3 we compare our best Li-GRU system with a more complex architecture based on a combination of feed-forward and recurrent models fed by a concatenation of features. To the best of our knowledge, the PER=13.8% achieved by the latter system yields the best-published performance on the TIMIT test-set.

Previous achievements were based on standard acoustic features computed with Kaldi. However, within PyTorch-Kaldi users can employ their own features. Table 4 shows the results achieved with convolutional models fed by standard FBANKs coefficients or by raw acoustic waveform directly. The standard CNN based on raw samples performs similarly to the one fed by FBANK features. A slight performance improvement is observed with SincNet [33], whose effectiveness to process raw waveforms for speech recognition is here highlighted for the first time.

We now extend our experimental validation to other datasets. With this regard, Table 5 shows the performance achieved on DIRHA, CHiME, and Librispeech (100h) datasets. The Table consistently show better performance with the Li-GRU model, confirming our previous achievements on TIMIT. The results on DIRHA and CHiME show the effectiveness of the proposed toolkit also in noisy condition. In particular, DIRHA represents a very challenging task, that is characterized by the presence of considerable levels of noise and reverberation. The WER=23.9% obtained on this dataset represents the best performance published so-far on the single-microphone task. Finally, the performance obtained with Librispeech outperforms the corresponding *p-norm* Kaldi baseline ($WER = 6.5\%$) on the considered 100 hours subset.

Table 5: WER(%) obtained for the DIRHA, CHiME, and LibriSpeech (100h) datasets with various neural architectures.

	DIRHA	CHiME	LibriSpeech
MLP	26.1	18.7	6.5
LSTM	24.8	15.5	6.4
GRU	24.8	15.2	6.3
Li-GRU	**23.9**	**14.6**	**6.2**

5. CONCLUSIONS

This paper described the PyTorch-Kaldi project, a new initiative that aims to bridge the gap between Kaldi and PyTorch. The toolkit is designed to make the development of an ASR system simpler and more flexible, allowing users to easily plug-in their customized acoustic models. PyTorch-Kaldi also supports combinations of neural architectures, features, and labels, allowing users to possibly employ complex ASR pipelines. The experiments have confirmed that PyTorch-Kaldi can achieve state-of-the-art results in some popular speech recognition tasks and datasets.

The current version of the PyTorch-Kaldi is already publicly-available along with a detailed documentation. The project is still in its initial phase and we invite all potential contributors to participate in it. We hope to build a community of developers larger enough to progressively maintain, improve, and expand the functionalities of our current toolkit. In the future, we plan to increase the number of pre-implemented models coded in our framework, and we would like to extend the current project by integrating neural language model training, as well proper support for end-to-end ASR systems.

6. ACKNOWLEDGMENT

We would like to thank Maurizio Omologo for his helpful comments. This research was enabled in part by support provided by Calcul Québec and Compute Canada.

7. REFERENCES

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

[2] M. Ravanelli, *Deep learning for Distant Speech Recognition*, PhD Thesis, Unitn, 2017.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[4] S. Renals, T. Hain, and H. Bourlard, "Interpretation of Multiparty Meetings the AMI and Amida Projects," in *Proc. of HSCMA*, 2008, pp. 115–118.

[5] M. Omologo, "A prototype of distant-talking interface for control of interactive TV," in *Proceedings of Asilomar Conference on Signals, Systems and Computers*, 2010.

[6] L. Cristoforetti, M. Ravanelli, M. Omologo, A. Sosi, A. Abad, M. Hagmueller, and P. Maragos, "The DIRHA simulated corpus," in *Proc. of LREC*, 2014, pp. 2629–2634.

[7] J. Barker, R. Marxer, E. Vincent, and S. Watanabe, "The third CHiME Speech Separation and Recognition Challenge: Dataset, task and baselines," in *Proc. of ASRU*, 2015, pp. 504–511.

[8] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *Proc. of ICASSP*, 2015, pp. 5206–5210.

[9] S. Young et al., *HTK – Hidden Markov Model Toolkit*, 2006.

[10] A. Lee and T. Kawahara., "Recent development of open-source speech recognition engine julius," in *Proc. of APSIPA-ASC*, 2008.

[11] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney, "RASR - The RWTH Aachen University Open Source Speech Recognition Toolkit," in *Proc. of ASRU*, 2011.

[12] G. Linarès, P. Nocera, D. Massonié, and D. Matrouf, "The lia speech recognition system: From 10xrt to 1xrt," in *Text, Speech and Dialogue*, Václav Matoušek and Pavel Mautner, Eds. 2007, pp. 302–308, Springer Berlin Heidelberg.

[13] D. Povey et al., "The Kaldi Speech Recognition Toolkit," in *Proc. of ASRU*, 2011.

[14] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

[15] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[16] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in *Proc. of USENIX-OSDI Symposium*, 2016, pp. 265–283.

[17] F. Seide and A. Agarwal, "CNTK: Microsoft's Open-Source Deep-Learning Toolkit," in *Proceedings of ACM SIGKDD*, 2016, pp. 2135–2135.

[18] A. Paszke et al., "Automatic differentiation in pytorch," 2017.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of ICML*, 2015, pp. 448–456.

[20] L. J. Ba, R. K., and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[22] D.P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of ICLR*, 2015.

[23] M. Ravanelli and M. Omologo, "Automatic context window composition for distant speech recognition," *Speech Communication*, vol. 101, pp. 34 – 44, 2018.

[24] M.J.F. Gales, "Maximum Likelihood Linear Transformations for HMM-Based Speech Recognition," *Computer Speech and Language*, vol. 12, no. 4, pp. 75–98, 1998.

[25] M. Ravanelli and M. Omologo, "Contaminated speech training methods for robust DNN-HMM distant speech recognition," in *Proc. of Interspeech*, 2015, pp. 756–760.

[26] P. Bell, P. Swietojanski, and S. Renals, "Multitask learning of context-dependent targets in deep neural network acoustic models," *IEEE/ACM Trans. Audio, Speech & Language Processing*, vol. 25, no. 2, pp. 238–247, 2017.

[27] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Batch-normalized joint training for dnn-based distant speech recognition," in *Proc. of SLT*, 2016, pp. 28–34.

[28] A. Narayanan and D. Wang, "Joint noise adaptive training for robust automatic speech recognition," in *Proc. of ICASSP*, 2014, pp. 4380–4384.

[29] X. Xiao et al., "Deep beamforming networks for multi-channel speech recognition," in *Proc. of ICASSP*, 2016.

[30] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "A network of deep neural networks for distant speech recognition," in *Proc. of ICASSP*, 2017, pp. 4880–4884.

[31] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Light gated recurrent units for speech recognition," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 2, pp. 92–102, April 2018.

[32] M. Ravanelli, D. Serdyuk, and Y. Bengio, "Twin regularization for online speech recognition," in *Proc. of Interspeech*, 2018.

[33] M. Ravanelli and Y.Bengio, "Speaker Recognition from raw waveform with SincNet," in *Proc. of SLT*, 2018.

[34] M. Ravanelli and Y.Bengio, "Interpretable Convolutional Filters with SincNet," in *Proc. of NIPS@IRASL*, 2018.

[35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[36] M. Ravanelli, L. Cristoforetti, R. Gretter, M. Pellin, A. Sosi, and M. Omologo, "The DIRHA-English corpus and related tasks for distant-speech recognition in domestic environments," in *Proc. of ASRU*, 2015, pp. 275–282.

[37] M. Ravanelli, P. Svaizer, and M. Omologo, "Realistic multi-microphone data simulation for distant speech recognition," in *Proc. of Interspeech*, 2016, pp. 2786–2790.

[38] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. of AISTATS*, 2010, pp. 249–256.

[39] Q.V. Le, N. Jaitly, and G.E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv:1504.00941*, 2015.

[40] T. Moon, H. Choi, H. Lee, and I. Song, "RNNDROP: A novel dropout for RNNS in ASR," in *Proc. of ASRU*, 2015.

[41] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *Proc. of ICASSP*, 2016, pp. 2657–2661.

[42] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Improving speech recognition by revising gated recurrent units," in *Proc. of Interspeech*, 2017.