Compressing Speech Recognition Networks with MLP via Tensor-Train Decomposition

Dan He[†‡] and Yubin Zhong[*]
[†] Guangzhou University, Guangzhou, China
[‡] CSLT, Tsinghua Unuiversity, China
E-mail: hedan@cslt.org
[*] Guangzhou University, Guangzhou, China
E-mail: Zhong_yb@163.com

***Abstract*—Deep neural networks (DNNs) have produced state-of-the-art performance in automatic speech recognition (ASR). This success is often associated with a large DNN structure with millions or even billions of parameters. Such large-scale networks take large disk space and require huge computational resources at run-time, therefore not suitable for applications in mobile or wearable devices. In this paper, we investigate a compression approach for DNNs based on Tensor-Train (TT) decomposition and apply it to the ASR task. Our results on the TIMIT database reveals that the compressed networks can maintain the performance of the original full-connected network, while greatly reducing the number of parameters. In particular, we found that the rate of model size decreasing is much larger than the rate of WER (word error rate) increasing, which means that the performance loss caused by the TT-based compression can be well compensated by the model size reduction. Moreover, how many layers and which layer can be substituted by TT is application dependent and should be carefully designed according to the application scenario.**

I. Introduction

Following years of research, deep learning has achieved state-of-the-art results on many sequential data modeling tasks, such as speech recognition [1, 2, 3, 4], machine translation [5], image classification [6][7] and object detection [8]. This brilliant success should be certainly attributed to the continuous effort of researchers on the design of various deep learning components, including network structures, training strategies and inference algorithms, but should be also attributed to the rapid growth in the amount of data available for training, and the great improvements in the computational power of hardware devices. The huge amount of data and computational power enables training large-sale deep neural network (DNN) that involve millions even billions of parameters [9].

The large-scale networks trained with large-scale data have been demonstrated to be very powerful. However, inferring with such a large structure requires intensive computational power and a large size of memory, which is only feasible for professional servers. If we want to deploy the model to mobile and wearable devices, the DNN structure should be significantly compressed. The necessity of this compression is mostly notable for automatic speech recognition (ASR) applications, as the pervasive deployment of wearable and IoT devices will require pervasive device-end ASR service, and the resources of these devices are very limited.

A prominent approach to DNN compression is matrix approximation. For example, the SVD algorithm can approximate a large matrix by a product of two low-rank matrices [10]. This approach has been successfully applied to compress pre-trained models. A related approach is to design structural matrices [11], where the parameters are shared according to pre-defined structural constraint. With this constraint, the free parameters are limited although the size of the matrices could be large. All these methods offer reasonable compression rate, but it still can not meet the requirement for large-scale DNN compression. Pruning method [12] is another idea for compress DNN networks, which cuts off those redundant and unimportant connections of the network and leave only the salient and essential structure.

Recently, the Tensor-Train (TT) decomposition [13] has attracted much attention. TT decomposition follows the same idea of matrix approximation and structural constraint, but the decomposition is by high-dimensional tensor product so the compression rate is much higher. In this paper, we conduct an empirical study of TT decomposition in speech recognition tasks. Our goal is train a very compact DNN acoustic model that can achieve comparable performance as a large-scale DNN. We found that TT decomposition works pretty well, and the performance loss imposed by this compression can by reasonably compensated for by the high compression rate, i.e., the decreasing in performance (in terms of WER) is much lower than the decreasing in the model size.

II. Related works

To meet the requirement of both state-of-the-art performance and efficient computation power and memory consumption, there is a trade-off between high-accuracy models and fast efficient models. A number of researchers have reported various methods aimed at minimizing the accuracy loss while maximizing the inference efficiency.

Courbariaux et al. [14] proposed binary-weight neural net, which significantly reduces the memory footprint and computational cost. The same ideas was also raised by Li et al. [15]. Hinton et al. [16] proposed a distillation approach that trains a

small network using soft targets generated by a large network, which can be regarded as a compression approach. Some researchers also tried to represent dense weight matrices with constrained structures. For example, Denil et al.[17] proposed low-rank matrix decomposition that replaces the original full-rank weight matrix with the product of two low-rank matrices.

Matrix decomposition is two-dimensional. Recently, this approach has been extended to tensor decomposition, which replaces the original full-rank weight matrix with the product of series of high-dimensional tensors. Due to the high dimensionality, the structure constraint in tensor decomposition is much richer than matrix decomposition. Among various tensor decomposition methods (e.g., CP decomposition and Tucker-decomposition), TT decomposition is the most successful due to its simplicity and flexibility [13, 18]. Recently, TT decomposition has been applied to many deep learning tasks. For example, Novikov et al. [19] used the TT format to represent the weight matrices in the fully connected layer inside a CNN model. Tjandra et al. [18] applied TT decomposition to compress the weight matrices inside RNN models. TT decomposition has also been applied to represent the embedding layer, a key ingredient in modern NLP [20].

Compared to other tensor decomposition methods, such as CP decomposition and Tucker-decomposition, TT decomposition tends to offer better performances [21]. For instance, the TT decomposition achieved the best performance when modeling polyphonic music [21]. In speech recognition, Mori et al. [22] applied TT decomposition on the end-to-end ASR framework. In our work, we will decompose fully-connected layers using TT on a simple phoneme recognition task, and investigate how to improve the performance of TT in such a classical configuration.

III. Tensor Train decomposition

In this section, we discuss the details of our proposed approach to compress DNN using the TT decomposition. The basic idea is to replace the dense matrix of a full-connected layer with a format of tensor train. This format is called TT format, and the layer in the TT format is called a TT layer.

A. Tensor-train (TT) format

We represent one-dimensional arrays as *vectors*, two-dimensional arrays as *matrices*, high-dimensional arrays as *tensors*. Bold lower case letters (e. g. $\boldsymbol{b}$) denote vectors; ordinary lower case letters (e. g. $b(i) = b_i$) denotes vector elements; bold upper case letters (e. g. $\boldsymbol{W}$) denote matrices; ordinary upper case letters (e. g. $W(i, j)$) denote matrix elements; calligraphic bold upper case letters (e. g. $\boldsymbol{\mathcal{W}}$) denote for tensors and ordinary calligraphic upper case letters (e. g. $\mathcal{W}(\boldsymbol{i}) = \mathcal{W}(i_1, \ldots, i_d)$) denote tensor elements, where d is the dimensionality of the tensor $\boldsymbol{\mathcal{W}}$. We will call arrays *explicit* to highlight cases when they are stored explicitly, i. e. by enumeration of all the elements.

A d-dimensional array (tensor) $\boldsymbol{\mathcal{W}}$ is said to be represented in the *TT-format* if for each dimension $k = 1, \ldots, d$ and for each possible value of the k-th dimension index $j_k = 1, \ldots, n_k$ there exists a matrix $\boldsymbol{G}_k[j_k]$ such that all the elements of $\boldsymbol{\mathcal{W}}$ can be computed as the following matrix product:

$$\mathcal{W}(j_1, \ldots, j_{d-1}, j_d) = \boldsymbol{G}_1[j_1]\boldsymbol{G}_2[j_2] \cdots \boldsymbol{G}_d[j_{d-1}]\boldsymbol{G}_d[j_d], \tag{1}$$

where matrices $\boldsymbol{G}_k[j_k]$ are related to the same dimension k, and they are restricted to be of the same size $\mathrm{r}_{k-1} \times \mathrm{r}_k$. The values r_0 and r_d are set to 1, so the final matrix product result is a scalar. We refer to the representation of a tensor in the TT-format as the *TT-decomposition*, a sequence $\{\mathrm{r}_k\}_{k=0}^{d}$ is referred to as the *TT-ranks* of the TT-representation of $\boldsymbol{\mathcal{W}}$, its maximum – as the *maximal TT-rank* of the TT-representation of $\boldsymbol{\mathcal{W}}$: $\mathrm{r} = \max_{k=0,\ldots,d} \mathrm{r}_k$. A collection of the matrices $(\boldsymbol{G}_k[j_k])_{j_k=1}^{n_k}$ corresponding to the same dimension are called the *TT-cores*.

We use the symbols $G_k[j_k](c_{k-1}, c_k)$ to denote the element of the matrix $\boldsymbol{G}_k[j_k]$ in the position (c_{k-1}, c_k), where $c_{k-1} = 1, \ldots, \mathrm{r}_{k-1}$, $c_k = 1, \ldots, \mathrm{r}_k$. Equation (1) can be equivalently rewritten as the sum of the products of the elements of the cores:

$$\mathcal{W}(j_1, \ldots, j_d) = \sum_{c_0, \ldots, c_d} G_1[j_1](c_0, c_1) \ldots G_d[j_d](c_{d-1}, c_d). \tag{2}$$

If a tensor $\boldsymbol{\mathcal{W}}$ is stored in the TT-format, in other words, using the *TT-cores* $\boldsymbol{G}_k[j_k]$ to represent the original tensor, the parameters that are truly stored in the memory will be reduced from $\prod_{k=1}^{d} n_k$ to $\sum_{k=1}^{d} n_k \, \mathrm{r}_{k-1} \, \mathrm{r}_k$. It can be seen that the TT-format is very efficient in terms of memory footprint if the ranks are small.

B. Compressing DNN with TT-format

In this section, we introduce the TT-layer that can be used to compose DNNs. In short, the TT-layer is a fully-connected layer with the weight matrix stored in the TT-format. A TT-DNN is a DNN that consists of one or more TT-layers.

More specifically, we replace a full-connected layer with a TT layer. A fully-connected layers involves a linear transformation on an N-dimensional input vector $\boldsymbol{x}$:

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{3}$$

where $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ is the weight matrix and $\boldsymbol{b} \in \mathbb{R}^M$ is the bias vector. Usually the weight matrix $\boldsymbol{W}$ contains a huge number of parameters, most of which are redundant and so can be compressed by TT decomposition and saved in the TT format, converting to a TT layer. In details, represent the transformation as a TT, and reshape the input vector $\boldsymbol{x}$ and output vector $\boldsymbol{y}$ to a d-dimensional tensor $\boldsymbol{\mathcal{X}}$ and a d-dimensional tensor $\boldsymbol{\mathcal{Y}}$. The linear transformation (3) is expressed in the tensor form:

$$\mathcal{Y}(i_1, \ldots, i_d) = \sum_{j_1, \ldots, j_d} \boldsymbol{G}_1[i_1, j_1] \ldots \boldsymbol{G}_d[i_d, j_d] \, \mathcal{X}(j_1, \ldots, j_d) + \mathcal{B}(i_1, \ldots, i_d). \tag{4}$$

Based on this tensor representation, conventional training strategies based on back-propagation can be employed to optimize the parameters $\boldsymbol{G}_k[j_k]$, and inference can be conducted simply by tensor production.

IV. Experiments

In this section, we present the data and the model architecture adopted in our experiments to evaluate different configurations of TT-DNNs.

A. Configurations

Our experiments were performed with the TIMIT corpus. The database contains a total of 6300 sentences, 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States. The associated phoneme recognition task was chosen in our experiments as the benchmark, where the data configuration follows the Kaldi s5 recipe [23]).

The PyTorch-Kaldi [24] toolbox was used to conduct the experiments, where PyTorch is responsible for NN training, while Kaldi is responsible for data processing and decoding. The architecture is shown in Fig. 1.

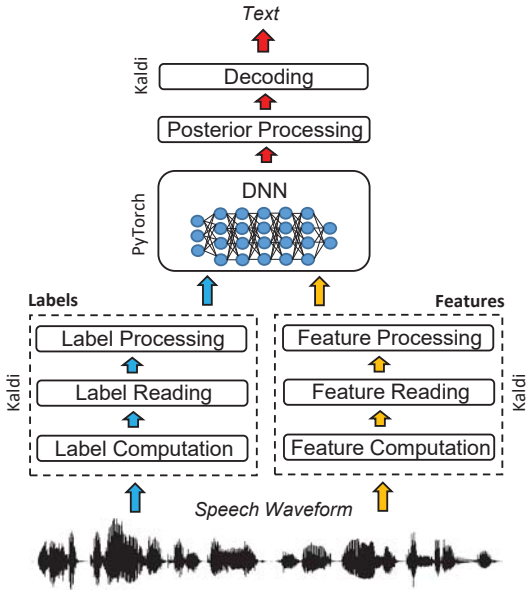

Fig. 1. An overview of the PyTorch-Kaldi architecture [24].

The baseline DNN model is a multi-layer perceptron (MLP) with 4 full-connected hidden layers. The size of the input layer is 429, corresponding to the dimensionality of the input feature, and the size of the output layer is 1952, corresponding to the pdfs of the speech recognition system. Each hidden layer contains 1024 units. All these layers can be formulated into a tensor form so that the TT decomposition can be employed. The details of the tensor format and the TT-rank of each layer are shown as follows.

1) Regular DNN
   - Input size: 429
   - Hidden-1 size : 1024
   - Hidden-2 size : 1024
   - Hidden-3 size : 1024
   - Hidden-4 size : 1024
   - output size :1952
2) Tensor-form DNN
   - Tensor input shape : $1 \times 3 \times 11 \times 13$
   - Tensor hidden shape : $4 \times 4 \times 8 \times 8$
   - Tensor output shape : $2 \times 2 \times 8 \times 61$
3) TT-rank
   - $1 \times 3 \times 3 \times 3 \times 1$

B. Single TT layer

The first experiment is to apply the TT decomposition to a particular layer, and compare the impact of the TT substitution on different layers, in terms of the word error rate (WER) and the model size.

We denote the number of layers that can be TT substituted as n, which is 5 in our experiments (the tensors that derive the hidden and output units), and the number of layers that has been TT substituted as m. In this experiment, only $m = 1$ is considered. To make the notation more clear, 'FC' is used to denote a full-connected layer, and 'TT' is used to denote a TT decomposed layer. An architecture 'TT-FC-FC-FC-FC' represents a TT-DNN whose layers are all full-connected, except the first hidden layer.

TABLE I
Comparison for different DNN/TT-DNN configurations ($m = 1$)

Architecture	Model size	WER
FC-FC-FC-FC-FC	21.4M	18.2%
TT-FC-FC-FC-FC	19.7M	18.2%
FC-TT-FC-FC-FC	17.4M (17894KB)	19.1%
FC-FC-TT-FC-FC	17.4M (17894KB)	18.7%
FC-FC-FC-TT-FC	17.4M (17894KB)	18.6%
FC-FC-FC-FC-TT	13.8M	19.0%
FC-FC-FC-FC	17.4M (17860KB)	37.5%

Table I shows the performance of different architectures. The 1st row shows the baseline, and the 2nd to 6th rows show the performance of TT-DNNs where the TT decomposition is applied to different layers. The 7th row reports the performance of another full-connected DNN model which contains three hidden layers (rather than four as in the baseline) and involves the same amount of parameters as the TT-DNNs.

We firstly observe that the TT decomposition can significantly reduce the size of a layer, leading to a drop of 1/5 in the number of model parameters. This parameter compression, however, does not lead to much significant performance reduction. The absolute WER increasing is controlled within 1%. In particular, when the TT decomposition is applied to the first hidden layer, there is no performance loss. Applying TT onto the second hidden layer and the output layer leads to the most significant performance reduction, though more investigation is required to analyze which layer is more amiable to TT decomposition.

Finally, when compared to the DNN model with 3 full-connected hidden layers (7th row), the TT-DNN models perform much better although the model sizes are almost the

same. This confirms that the TT format is an elegant structure that can learn important information with very limited amount of parameters.

C. Two TT layers

TABLE II
COMPARISON FOR DIFFERENT DNN/TT-DNN CONFIGURATIONS ($m = 2$)

Architecture	Model size	WER
FC-FC-FC-FC-FC	21.4M	18.2%
TT-TT-FC-FC-FC	15.8M	18.8%
TT-FC-TT-FC-FC	15.8M	19.2%
TT-FC-FC-TT-FC	15.8M	19.0%
TT-FC-FC-FC-TT	12.1M	19.5%
FC-TT-TT-FC-FC	13.4M	20.1%
FC-TT-FC-TT-FC	13.4M	19.7%
FC-TT-FC-FC-TT	9.85M	21.1%
FC-FC-TT-TT-FC	13.4M	19.1%
FC-FC-TT-FC-TT	9.85M	20.1%
FC-FC-FC-TT-TT	9.85M	20.7%
FC-FC-FC	13.4M	40.4%

The results of TT-DNNs with 2 TT layers are shown in Table II. It can be seen that with the additional TT layer, the model size is further compressed, with noticeable but limited performance reduction. When compared with a DNN model with 2 full-connected hidden layers (the last row), the TT-DNNs with 2 TT layers perform much better although their sizes are the similar. All these observations are consistent with the results in Table I.

D. Mutiple TT layers

TABLE III
COMPARISON OF DIFFERENT DNN/TT-DNN CONFIGURATIONS ($m = 3, 4, 5$)

Architecture	Model size	WER
FC-FC-FC-FC-FC	21.4M	18.2%
FC-TT-TT-TT-FC	9.48M	21.1%
TT-TT-TT-TT-FC	7.81M	23.1%
TT-TT-TT-TT-TT	0.2M	52.2 %
FC-FC	9.38M	46.1%

The results with more TT layers are shown in Table III. It can be seen that with more hidden layers formulated by TT, the model size keeps decreased, and the performance keeps degraded, but still under control. However, it seems converting the output layer to be TT is risky: although the model size can be significantly reduced to 200k bytes, the performance is dropped to an unacceptable level. This indicates the TT format is compact but less flexible, i.e., the functional space it can cover is limited. Therefore, TT layers should collaborate with some more general functions (e.g., full-connected layers) to achieve the learning goal.

Plot all the experimental results on a two-dimensional space where the x-axis is the compression ratio and the y-axis is the WER, and then conduct a polynomial regression. The results are shown in Fig. 2. It can be seen that when the compression ratio is below 60%, the performance will be not impacted much and the WER will keep stable. However, when the model is compressed more aggressively, the WER is significantly increased and the model becomes unacceptable. In practice, the trade-off between compression ratio and performance loss needs to carefully addressed.

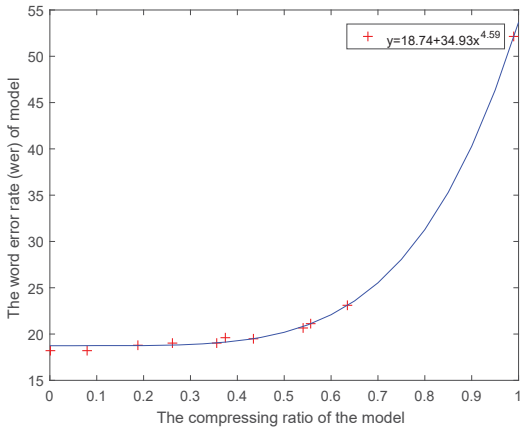

Fig. 2. Model compressing ratio versus WER with TT-DNNs. The function in the legend is fitted using the standard least-square method.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated a model compression approach for speech recognition, by using the tensor train (TT) format. This format introduces complex relationships among parameters via tensor multiplication, leading to structural priors that make it possible to encode complex patterns with limited parameters, hence a compact model.

The experiments were conducted on a phoneme recognition task with the TIMIT database. The results show that the TT formulation can greatly reduce the model size, with little performance loss. With more TT layers involved, the model size keeps decreased, and the performance keeps degraded but still under control. However, if all the layers are TT, the performance will be significantly reduced, indicating that pure TT models are not powerful enough and so require at least one full-connected layer to deal with the complex patterns of speech signals. Interestingly, TT-DNNs perform much better than full-connected DNNs with a similar model size.

In the future, we will study the theoretical limitation of the TT format and investigate methods to improve its capacity. We will also study the method with larger datasets.

VI. ACKNOWLEDGMENT

This work was conducted when the first author was an intern student at the Center for Speech and Language Technology (CSLT), Tsinghua University. Thanks to the CSLT people for the valuable discussions and suggestions, in particular Dr. Zhiyuan Tang, Dr. Lantian Li. The work was partially supported by the National Natural Science Foundation of China under Projects 61633013.

REFERENCES

[1] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6645–6649.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al., "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, 2016, pp. 173–182.

[3] Zhiyuan Tang, Ying Shi, Dong Wang, Yang Feng, and Shiyue Zhang, "Memory visualization for gated recurrent neural networks in speech recognition," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2736–2740.

[4] Dong Wang, Qiang Zhou, and Amir Hussain, "Deep and sparse learning in speech and language processing: An overview," in *International Conference on Brain Inspired Cognitive Systems*. Springer, 2016, pp. 171–183.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] Dan Kalman, "A singularly valuable decomposition: the svd of a matrix," *The college mathematics journal*, vol. 27, no. 1, pp. 2–23, 1996.

[11] Tyson R Browning, "Applying the design structure matrix to system decomposition and integration problems: a review and new directions," *IEEE Transactions on Engineering management*, vol. 48, no. 3, pp. 292–306, 2001.

[12] Chao Liu, Zhiyong Zhang, and Dong Wang, "Pruning deep neural networks by optimal brain damage," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[13] Ivan V Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

[14] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[15] Lantian Li, Chao Xing, Dong Wang, Kaimin Yu, and Thomas Fang Zheng, "Binary speaker embedding," in *2016 10th International Symposium on Chinese Spoken Language Processing (ISCSLP)*. IEEE, 2016, pp. 1–4.

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[17] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al., "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[18] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura, "Compressing recurrent neural network with tensor train," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 4451–4458.

[19] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov, "Tensorizing neural networks," in *Advances in neural information processing systems*, 2015, pp. 442–450.

[20] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets, "Tensorized embedding layers for efficient model compression," *arXiv preprint arXiv:1901.10787*, 2019.

[21] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura, "Tensor decomposition for compressing recurrent neural network," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[22] Takuma Mori, Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura, "Compressing end-to-end asr networks by tensor-train decomposition.," in *Interspeech*, 2018, pp. 806–810.

[23] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number CONF.

[24] Mirco Ravanelli, Titouan Parcollet, and Yoshua Bengio, "The pytorch-kaldi speech recognition toolkit," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6465–6469.