Interspeech 2018
2-6 September 2018, Hyderabad

Compressing End-to-end ASR Networks by Tensor-Train Decomposition

Takuma Mori[1], *Andros Tjandra*[1,2], *Sakriani Sakti*[1,2], *Satoshi Nakamura*[1,2]

[1]Graduate School of Information Science, Nara Institute of Science and Technology, Japan
[2]RIKEN, Center for Advanced Intelligence Project AIP, Japan

{mori.takuma.mi1, andros.tjandra.ai6, ssakti, s-nakamura}@is.naist.jp

Abstract

End-to-end deep learning has become a popular framework for automatic speech recognition (ASR) tasks, and it has proven itself to be a powerful solution. Unfortunately, network structures commonly have millions of parameters, and large computational resources are required to make this approach feasible for training and running such networks. Moreover, many applications still prefer lightweight models of ASR that can run efficiently on mobile or wearable devices. To address this challenge, we propose an approach that can reduce the number of ASR parameters. Specifically, we perform Tensor-Train decomposition on the weight matrix of the gated recurrent unit (TT-GRU) in the end-to-end ASR framework. Experimental results on LibriSpeech data reveal that the compressed ASR with TT-GRU can maintain good performance while greatly reducing the number of parameters.

Index Terms: tensor train decomposition, compression technique, end-to-end model, automatic speech recognition

1. Introduction

Speech recognition has certainly come a long way, from a simple machine that responds to a small set of sounds to highly sophisticated systems that recognize conversational speech. One of the fastest growth areas is the market for mobile phones and wearable devices. Currently, the state-of-the-art ASR system has already reached a level that enables the user to conduct dialogues with computerized personal assistants (i.e., Apples Siri, Amazons Alexa, Google Now, and Microsofts Cortana).

ASR technologies have also made remarkable progress. Traditional ASR typically performs multi-level pattern recognition tasks based on hidden Markov models (HMM) that map the acoustic speech waveform into a hierarchy of speech units such as sub-words (phonemes), words, and strings of words (sentences). Recently, end-to-end deep learning has become a popular framework for ASR tasks and has proven itself powerful [1, 2, 3]. One of the important factors behind the popularity of deep learning is the possibility of simplifying many complicated hand-engineered models by letting DNNs find their own way in mapping from input to output spaces.

However, the overall network structure commonly consists of multiple feed-forward and recurrent hidden layers. Most recurrent neural network (RNN) models are computationally expensive and have a huge number of parameters. Consequently, large computational resources are required to make the system feasible for training and running such networks. This limitation can hinder us in constructing deep learning based ASR systems that can be fast enough for massive real-time inference or small enough to be implemented in low-end devices like mobile phones [4] and embedded systems with limited available memory.

To address this challenge, we propose an approach that could reduce the number of ASR parameters. Specifically, we perform Tensor-Train decomposition on the weight matrix of the gated recurrent unit (TT-GRU) in the end-to-end ASR framework.

2. Related Works

To bridge the gap between high-performance state-of-the-art models and systems running under efficient computational and memory costs, 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 model's efficiency.

Li et al. proposed learning a DNN with a small number of hidden nodes by minimizing the Kullback-Leibler (KL) divergence between the output distributions of the small-sized DNN and a standard large-sized DNN [5]. Hinton et al. [6] successfully compressed a large deep neural network into a smaller neural network by training the smaller neural network on the transformed softmax outputs from the vast deep neural network. Distilling knowledge from the larger neural network has also been applied to recurrent neural network architecture by Tang et al. [7]. Another approach by Denil et al. [8] utilized low-rank matrix decomposition of the weight matrices. A Study by Ba et al. [9] demonstrated that it is possible to train shallow neural nets to mimic deep models and perform similarly to well-engineered complex deep convolutional architectures.

Moreover, Sainath et al. showed that parameters could be reduced by low-rank matrix decomposition of the final layer [10]. Xue at al. proposed reducing the model size while maintaining improvements in accuracy by applying singular value decomposition (SVD) to the weight matrices in DNN and then restructuring the model based on the inherent sparseness of the original matrices [11]. Denton et al. also showed that SVD of the filter of the convolution neural net could reduce parameters and speed up the setting [12]. Wang and colleagues succeeded in compressing the entire coupling layer by combining SVD and vector quantization to compress the acoustic model [13]. Prabhavalkar et al. proposed a general recurrent model compression that jointly compresses both recurrent and non-recurrent inter-layer weight matrices. The proposed technique could reduce the model size to a third of its original size [14].

A recent study by Novikov et al. [15] replaced the dense weight matrices with Tensor Train (TT) format [16] inside a convolutional neural network (CNN) model. By using TT-format, they manage to compress the number of parameters significantly and keep the model accuracy degradation as low as possible. Tjandra et al. then showed that the parameters of a neural network can be decomposed and compressed by tensor train in MIDI music classification [17, 18]. However, to the best of our knowledge, no study has focused on compressing more complex neural networks with tensor-based representation for ASR tasks.

3. End-to-end ASR System

Our ASR system is built upon "Deep Speech 2" architecture. It consists of multiple layers including many bidirectional GRU layers and convolutional layers. Here, we perform Tensor-Train decomposition on the weight matrix of the gated recurrent unit (TT-GRU). Figure 1 illustrates the modification that was applied to the basic architecture of Deep Speech 2 (left side) in the proposed model (right side).

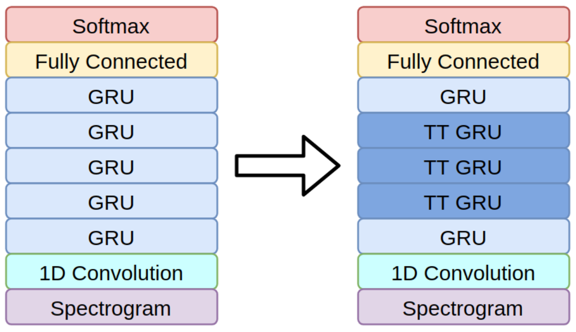

Figure 1: *The basic architecture of Deep Speech 2 (left side) and the proposed model (right side).*

4. Tensor-Train Decomposition

In this section, we describe the details of our proposed approach to compress GRU using the TT-format representation.

4.1. Tensor-Train (TT) format

We represent vectors with lowercase letters (e.g b), matrices with uppercase letters (e.g W), and tensors with calligraphic uppercase letters (e.g $\mathcal{W}$). Each element from the vectors, matrices and tensors is represented explicitly using indexing in every dimension. For example: $b(i)$ is the i-th element from vector b, $W(p,q)$ is the element of the p-th row and q-th column from matrix W, and $\mathcal{W}(j_1,..,j_d)$ is the element at index $(j_1,..,j_d)$ of tensor $\mathcal{W}$ with d being the order of tensor $\mathcal{W}$.

Based on the description in [15], we can assume that the d-dimensional array (tensor) $\mathcal{W}$ is represented in TT format [16] if for each $k \in \{1,..,d\}$ and for each possible value of the k-th dimension index $j_k \in \{1,..,n_k\}$ there exists a matrix $G_k[j_k]$ such that all elements of $\mathcal{W}$ can be computed as the following equation:

$$\mathcal{W}(j_1,j_2,..,j_{d-1},j_d) = G_1[j_1]\cdot G_2[j_2]...G_{d-1}[j_{d-1}]\cdot G_d[j_d]. \quad (1)$$

For all matrices $G_k[j_k]$ related to the same dimension k, they must be represented with size $r_{k-1}\times r_k$, where r_0 and r_d must be equal to 1 to retain the final matrix multiplication result as a scalar.

In TT-format, we define a sequence of rank $\{r_k\}_{k=0}^{d}$ and call them TT-rank from tensor $\mathcal{W}$. The set of matrices $\mathcal{G}_k = \{G_k[j_k]\}_{j_k=1}^{n_k}$, where the matrices are spanned in the same index, is called TT-core. We can describe Equation 1 in detail by enumerating the indices $q_{k-1} \in \{1,..,r_{k-1}\}$ and $q_k \in \{1,..,r_k\}$ in matrix $G_k[j_k]$ across all $k \in \{1,..,d\}$:

$$\mathcal{W}(j_1,j_2,..,j_{d-1},j_d) = \sum_{q_0,..,q_d} G_1[j_1](q_0,q_1)..G_d[j_d](q_{d-1},q_d). \quad (2)$$

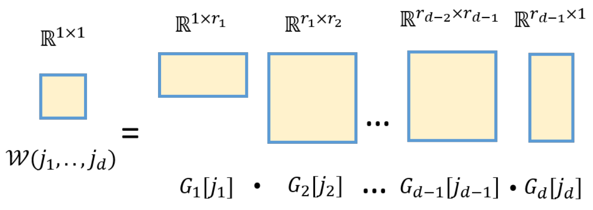
$\mathbb{R}^{1\times 1}$
$\mathbb{R}^{1\times r_1}$
$\mathbb{R}^{r_1\times r_2}$
$\mathbb{R}^{r_{d-2}\times r_{d-1}}$
$\mathbb{R}^{r_{d-1}\times 1}$
$\mathcal{W}(j_1,..,j_d)$
=
...
$G_1[j_1]$ • $G_2[j_2]$ ... $G_{d-1}[j_{d-1}]$ • $G_d[j_d]$

Figure 2: *Illustration for Equation 1: Calculating an element $\mathcal{W}(j_1,..,j_k)$ using the set of TT-cores $\{G_k[j_k]\}_{k=1}^{d}$*

.

By factoring the original tensor $\mathcal{W}$ into multiple TT-cores $\{\mathcal{G}_k\}_{k=1}^{d}$, we are able to compress the number of elements needed to represent the original tensor size from $\prod_{k=1}^{d} n_k$ to $\sum_{k=1}^{d} n_k r_{k-1} r_k$.

4.2. Representing Linear Transformation using TT-format

Nearly all parts of neural networks are composed of linear transformations:

$$y = Wx + b, \quad (3)$$

where $W \in \mathbb{R}^{M\times N}$ is the weight matrix and $b \in \mathbb{R}^M$ is the bias vector. In most cases, matrix W has a much larger number of parameters compared to the bias b. Therefore, we are able to utilize TT-format in optimizing our neural networks by replacing the weight matrix W with tensor $\mathcal{W}$ in TT-format [15].

We represent TT-format for matrix $W \in \mathbb{R}^{M\times N}$, where $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$, as tensor $\mathcal{W}$ by defining bijective functions $\mathbf{f}_i : \mathbb{Z}_+ \to \mathbb{Z}_+^d$ and $\mathbf{f}_j : \mathbb{Z}_+ \to \mathbb{Z}_+^d$. Function $\mathbf{f}_i$ maps each row $p \in \{1,..,M\}$ into $\mathbf{f}_i(p) = [i_1,..,i_d]$, and $\mathbf{f}_j$ maps each column $q \in \{1,..,N\}$ into $\mathbf{f}_j(q) = [j_1(q),..,j_d(q)]$. After we define such bijective functions, we can access the value from matrix $W(p,q)$ in tensor $\mathcal{W}$ with the index vectors generated by $\mathbf{f}_i(p)$ and $\mathbf{f}_j(q)$. We transform Eq. 1 to represent a matrix W in TT-format into:

$$\begin{aligned} W(p,q) &= \mathcal{W}(\mathbf{f}_i(p),\mathbf{f}_j(q)), & (4)\\ &= \mathcal{W}\left([i_1(p),..,i_d(p)],[j_1(q),..,j_d(q)]\right), & (5)\\ &= G_1\left[i_1(p),j_1(q)\right]..G_d\left[i_d(p),j_d(q)\right], & (6)\end{aligned}$$

where for each $k \in \{1,..,d\}$:

$$\begin{aligned} G_k[i_k(p),j_k(q)] &\in \mathbb{R}^{r_{k-1}\times r_k},\\ i_k(p) &\in \{1,..,m_k\},\\ j_k(q) &\in \{1,..,n_k\}.\end{aligned}$$

To represent linear transformation in Equation 3 with Equation 4-6, we need to reshape the vector input x into tensor $\mathcal{X}$ and the bias vector b into tensor $\mathcal{B}$ with order d to match our tensor $\mathcal{W}$. The following equation calculates a similar operation to $y(p) = W(p,:)x + b$, where we map row index p to vector $[i_1(p),..,i_d(p)]$ and enumerate all possible mappings for all columns in matrix W:

$$\mathcal{Y}\left(i_1(p),..,i_d(p)\right) = \sum_{j_1,..,j_d} G_1[i_1(p),j_1]..G_d[i_d(p),j_d]\cdot \mathcal{X}\left(j_1,..,j_d\right) + \mathcal{B}\left(i_1(p),..,i_d(p)\right). \quad (7)$$

Table 1: *Fully Connected vs TT Layer: Running Time and Memory*

Operation	Time	Memory
FC forward	$O(MN)$	$O(MN)$
TT forward	$O(dr^2 m \max(M,N))$	$O(r \max(M,N))$
FC backward	$O(MN)$	$O(MN)$
TT backward	$O(d^2 r^4 m \max(M,N))$	$O(r^3 \max(M,N))$

We can control the shape of TT-cores $\{\mathcal{G}_k\}_{i=1}^d$ by choosing the factor M as $\{m_k\}_{k=1}^d$ and N as $\{n_k\}_{k=1}^d$ as long as the number of factors are equal between M and N. We can also define the TT-rank $\{r_k\}_{k=0}^d$ and treat it as a hyper-parameter. In general, if we use a smaller TT-rank, we get more efficient models but also restrict the model's ability to learn more complex representation. If we use a larger TT-rank, we get more flexibility to express our weight parameters but we sacrifice our model's efficiency. Table 1 compares the forward and backward propagation time and memory complexity between the fully connected layer and the TT layer in Big-O notation [15]. We compare the fully connected layer having matrix $W \in \mathbb{R}^{M\times N}$ with the TT layer having tensor $\mathcal{W}$ and TT-rank $\{r_k\}_{k=0}^d$. In the table, m denotes $\max(\{m_k\}_{k=1}^d)$ and r denotes $\max(\{r_k\}_{k=0}^d)$.

4.3. Compressing GRU with TT-format

As illustrated in Figure 3, the GRU hidden layer at time t is defined by the following equations [19]:

$$\begin{aligned} r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r), & (8)\\ z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z), & (9)\\ \tilde{h}_t &= f(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h), & (10)\\ h_t &= (1-z_t)\odot h_{t-1} + z_t \odot \tilde{h}_t, & (11) \end{aligned}$$

where $\sigma(\cdot)$ is a sigmoid activation function, $f(\cdot)$ is the tanh activation function, r_t, z_t are the reset and update gates, $\tilde{h}_t$ is the candidate hidden layer values, and h_t is the hidden layer value at time-t. The reset gates control which previous hidden layer values are useful for the current candidate hidden layer. The update gates control the decision on whether to keep the previous hidden layer values or to replace the current hidden layer values with the candidate hidden layer values. GRU can match LSTM's performance, and its convergence speed sometimes surpasses that of LSTM despite having one fewer gating layer [20].

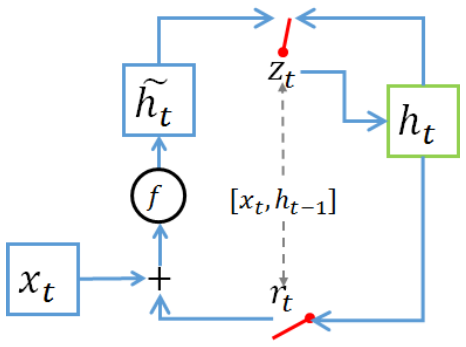

Figure 3: *Gated Recurrent Unit*

In this section, we apply TT-format to represent gated RNN. Among several RNN architectures with a gating mechanism, we chose GRU to be reformulated in TT-format because GRU has a less complex formulation but similar performance compared to LSTM. We call this model TT-GRU in this paper. Here, we focus our attention on six dense weight matrices (W_{xr}, W_{hr}, W_{xz}, W_{hz}, W_{xh}, W_{hh}). The weight matrices W_{xr}, W_{xz}, W_{xh} $\in \mathbb{R}^{M\times N}$ are parameters for projecting the input layer to the reset gate, update gate, and candidate hidden layer, respectively; W_{hr}, W_{hz}, $W_{hh} \in \mathbb{R}^{M\times M}$ are parameters for projecting the previous hidden layer to the reset gate, update gate, and candidate hidden layer, respectively.

We factorize $M = \prod_{k=1}^d m_k$, $N = \prod_{k=1}^d n_k$ and set TT-rank as $\{r_k\}_{k=0}^d$. All weight matrices (W_{xr}, W_{hr}, W_{xz}, W_{hz}, W_{xh}, W_{hh}) are substituted with tensor ($\mathcal{W}_{xr}$, $\mathcal{W}_{hr}$, $\mathcal{W}_{xz}$, $\mathcal{W}_{hz}$, $\mathcal{W}_{xh}$, $\mathcal{W}_{hh}$) in TT-format. Tensor $\mathcal{W}_{xr}$, $\mathcal{W}_{xz}$, $\mathcal{W}_{xh}$ are represented by a set of TT-cores ($\{\mathcal{G}_k^{xr}\}_{k=1}^d$, $\{\mathcal{G}_k^{xz}\}_{k=1}^d$, $\{\mathcal{G}_k^{xh}\}_{k=1}^d$), where $\forall k \in \{1,..,d\}$, $(\mathcal{G}_k^{xr}, \mathcal{G}_k^{xz}, \mathcal{G}_k^{xh} \in \mathbb{R}^{m_k \times n_k \times r_{k-1}\times r_k})$. Tensor $\mathcal{W}_{hr}$, $\mathcal{W}_{hz}$, $\mathcal{W}_{hh}$ are represented by a set of TT-cores ($\{\mathcal{G}_k^{hr}\}_{k=1}^d$, $\{\mathcal{G}_k^{hz}\}_{k=1}^d$, $\{\mathcal{G}_k^{hh}\}_{k=1}^d$), where $\forall k \in \{1,..,d\}$, $(\mathcal{G}_k^{hr}, \mathcal{G}_k^{hz}, \mathcal{G}_k^{hh} \in \mathbb{R}^{m_k\times m_k\times r_{k-1}\times r_k})$. We define bijective function $\mathbf{f}_i^x$ to access row p from W_{xr}, W_{xz}, W_{xh} and function $\mathbf{f}_i^h$ to access row p from W_{hr}, W_{hz}, W_{hh} in the set of TT-cores. We rewrite the GRU formulation to calculate r_t in Equation 8 as

$$\begin{aligned} a_t^{xr}(p) &= \sum_{j_1,\cdots,j_d} \mathcal{W}_{xr}(\mathbf{f}_i^x(p),[j_1,..,j_d]) \cdot \mathcal{X}_t\,(j_1,..,j_d)\,,\\ a_t^{hr}(p) &= \sum_{j_1,\cdots,j_d} \mathcal{W}_{hr}(\mathbf{f}_i^h(p),[j_1,..,j_d]) \cdot \mathcal{H}_{t-1}\,(j_1,..,j_d)\,,\\ a_t^{xr} &= [a_t^{xr}(1),..,a_t^{xr}(M)]\,,\\ a_t^{hr} &= \left[a_t^{hr}(1),..,a_t^{hr}(M)\right],\\ r_t &= \sigma(a_t^{xr} + a_t^{hr} + b_r). \qquad (12) \end{aligned}$$

Next, we rewrite the GRU formulation to calculate z_t in Equation 9 as

$$\begin{aligned} a_t^{xz}(p) &= \sum_{j_1,\cdots,j_d} \mathcal{W}_{xz}(\mathbf{f}_i^x(p),[j_1,..,j_d]) \cdot \mathcal{X}_t\,(j_1,..,j_d)\,,\\ a_t^{hz}(p) &= \sum_{j_1,\cdots,j_d} \mathcal{W}_{hz}(\mathbf{f}_i^h(p),[j_1,..,j_d]) \cdot \mathcal{H}_{t-1}\,(j_1,..,j_d)\,,\\ a_t^{xz} &= [a_t^{xz}(1),..,a_t^{xz}(M)]\,,\\ a_t^{hz} &= \left[a_t^{hz}(1),..,a_t^{hz}(M)\right],\\ z_t &= \sigma(a_t^{xz} + a_t^{hz} + b_z). \qquad (13) \end{aligned}$$

Finally, we rewrite the GRU formulation to calculate $\tilde{h}_t$ in Equation 10 as

$$\begin{aligned} a_t^{xh}(p) &= \sum_{j_1,\cdots,j_d} \mathcal{W}_{xh}(\mathbf{f}_i^x(p),[j_1,..,j_d]) \cdot \mathcal{X}_t\,(j_1,..,j_d)\,,\\ a_t^{hh}(p) &= \sum_{j_1,\cdots,j_d} \mathcal{W}_{hh}(\mathbf{f}_i^h(p),[j_1,..,j_d]) \cdot,\\ &\qquad (\mathcal{R}_t\,(j_1,..,j_d)\cdot \mathcal{H}_{t-1}\,(j_1,..,j_d))\,,\\ a_t^{xh} &= \left[a_t^{xh}(1),..,a_t^{xh}(M)\right],\\ a_t^{hh} &= \left[a_t^{hh}(1),..,a_t^{hh}(M)\right],\\ \tilde{h}_t &= f(a_t^{xh} + a_t^{hh} + b_h). \qquad (14) \end{aligned}$$

After all r_t, z_t and $\tilde{h}_t$ are calculated, we can calculate h_t in Equation 11 with standard operations like element-wise sum and multiplication.

In practice, we could assign different d for each weight tensor as long as the input data dimension could also be factorized into a d value. We could also assign a different TT-rank for each tensor and treat this as our model hyper-parameter. However, to simplify our implementation, we use the same TT-rank for both input projection weight tensor and hidden projection weight tensor. We also use the same factorization $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$ for all weight tensors in TT-GRU.

We do not substitute the bias vector b into tensor $\mathcal{B}$ because the number of bias parameters is insignificant compared to the number of parameters in matrix W. In terms of performance, the element-wise sum operation for bias vector b is also insignificant compared to matrix multiplication between a weight matrix and the input layer or previous hidden layer.

4.4. Initialization for TT-core parameters

Weight initialization is one of the most important tasks for training deep neural networks. Especially for our RNN with TT-format, which has many mini-tensors and several multiplications, the TT-RNN will have a longer matrix multiplication chain compared to a standard RNN, and the hidden layer value will quickly saturate [21]. Therefore, we need to choose the initialization method carefully so that our proposed model starts under a stable condition. In our implementation, we follow Glorot initialization [21] to keep the same variance of the weights' gradient across layers and time-steps to avoid the vanishing gradient problem. We initialize every TT-core as follows:

$$\begin{aligned} \forall k \in \{1,..,d\}, \quad \mathcal{G}_k \quad &\sim \quad \mathcal{N}(0, \sigma_k), \\ \text{where} \quad \sigma_k \quad &= \quad \sqrt{\frac{2}{(n_k \cdot r_k) + (m_k \cdot r_{k-1})}}. \end{aligned}$$

By choosing a good initialization, our neural network will be able to converge faster and obtain better local minima. Based on our preliminary experiments, we obtain better starting loss at the first several epochs compared to the randomly initialized model with the same σ_k on a Gaussian distribution for all TT-cores.

5. Experimental Set-up

The English speech corpus LibriSpeech corpus [22] is used in this study as a task to evaluate our proposed model. Due to time constraints, we only used the smallest "train-clean-100" subset for training data, "dev-clean" subset for validation data, and "test-clean" subset for test data as shown in Table 2.

Table 2: *Libri Speech data [22]*

subset	hours	speakers	per-spk minutes
train-clean-100	100.6	251	25
dev-clean	5h	40	10
test-clean	5h	40	10

The speech utterances were segmented into multiple frames with a 25-ms window size and a 10-ms step size. Then we extracted 23-dimension filter bank features using Kaldis feature extractor [23] and normalized them to have zero mean and unit variance.

The setting of the model parameters of the GRU baseline is based on the paper of Deep Speech 2 [3], and the configuration of the parameters of TT-GRU is determined based on the tensor decomposition. For baseline GRU, we used the SGD algorithm, since it provides the best performance. However, our TT-GRU could not converge with SGD, and thus we used the Adam algorithm to optimize the TT-GRU model parameters. We utilized TT-GRU implementation from `https://github.com/androstj/tensor_rnn`.

6. Experiment Results

Table 3 shows the performance of the proposed TT-GRU in comparison with the baseline (uncompressed) GRU. Here, GRU is denoted as "GRU-HF," and TT-GRU is denoted as "TT-GRU-HF-R," where "H" means "hidden units," "F" is the number of hidden units, and "R" represents tensor-train rank. For example, "GRU-H1510" means a GRU with 1510 hidden units, and "TT-GRU-H4x8x6x8-R3" means a TT-GRU with a hidden unit of 4x8x6x8 in the tensor-train rank 3 of the tensor-train format.

Table 3: *Speech recognition performance of baseline GRU and proposed TT-GRU.*

Model	Param	Comp	Val CER	Test CER
GRU-H1510	13M	100	20.03%	20.62%
TT-GRU				
H4x8x6x8-R3	11k	0.08	27.57%	27.21%
H4x8x6x8-R5	22k	0.16	23.76%	23.40%
H4x8x6x8-R7	37k	0.27	22.68%	23.73%

The baseline model is a GRU with 1510 hidden units. Our proposed model has a 4x8x6x8 output shape and TT-GRU of tensor train rank (3, 5, 7). No language model was applied, and thus a character error rate (CER) was used for the evaluation function. All of these models have similar performances based on the negative log-likelihood and accuracy in the test set. However, the best system of our proposed model could drastically reduce the parameters from 13M to 37k. Overall, this reduction was about 99% in the converted GRU layer and about 60% in the entire model. This reveals that the performance could be maintained while reducing the number of parameters.

Importantly, unlike several published systems using these benchmarks, our proposed system does not involve a language model. Therefore, the results reported in the paper could not reach state-of-the-art performance. Nevertheless, the results are still convincing as evidence of the proposed framework's effectiveness.

7. Conclusions

In this paper, we demonstrated an efficient and compact neural network model using TT-format representation for a CTC-based end-to-end automatic speech recognition task. By using TT-format, we were able to represent dense weight matrices inside the RNN layer with multiple low-rank tensors. We evaluated our proposed model on LibriSpeech data. Our proposed TT-GRU is able to compress the number of parameters significantly while retaining high model performance and accuracy at the same time. In the future, we will elaborate the TT-format in other parts of the network. Furthermore, we will incorporate a language model on top of the CTC layer of the proposed model and evaluate performance using a larger dataset.

8. Acknowledgements

Part of this work was supported by JSPS KAKENHI Grant Numbers JP17H06101 and JP17K00237.

9. References

[1] A. Graves and N. Jaitly, “Towards end-to-end speech recognition with recurrent neural networks,” in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1764–1772.

[2] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, “Deep speech: Scaling up end-to-end speech recognition,” *arXiv preprint arXiv:1412.5567*, 2014.

[3] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. 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.

[4] M. Schuster, “Speech recognition for mobile devices at Google,” in *Pacific Rim International Conference on Artificial Intelligence*. Springer, 2010, pp. 8–10.

[5] J. Li, R. Zhao, J.-T. Huang, and Y. Gong, “Learning small-size DNN with output-distribution-based criteria,” in *Interspeech*, September 2014.

[6] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, 2015.

[7] Z. Tang, D. Wang, and Z. Zhang, “Recurrent neural network training with dark knowledge transfer,” in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5900–5904.

[8] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, “Predicting parameters in deep learning,” in *Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[9] J. Ba and R. Caruana, “Do deep nets really need to be deep?” in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[10] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, “Low-rank matrix factorization for deep neural network training with high-dimensional output targets,” in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 6655–6659.

[11] J. Xue, J. Li, and Y. Gong, “Restructuring of deep neural network acoustic models with singular value decomposition,” in *INTERSPEECH*, 2013, pp. 2365–2369.

[12] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” in *Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[13] Y. Wang, J. Li, and Y. Gong, “Small-footprint high-performance deep neural network-based speech recognition using split-VQ,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4984–4988.

[14] R. Prabhavalkar, O. Alsharif, A. Bruguier, and L. McGraw, “On the compression of recurrent neural networks with an application to LVCSR acoustic modeling for embedded speech recognition,” in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 5970–5974.

[15] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, “Tensorizing neural networks,” in *Advances in Neural Information Processing Systems*, 2015, pp. 442–450.

[16] I. V. Oseledets, “Tensor-train decomposition,” *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011. [Online]. Available: http://dx.doi.org/10.1137/090752286

[17] A. Tjandra, S. Sakti, and S. Nakamura, “Compressing recurrent neural network with tensor train,” *International Joint Conference on Neural Networks (IJCNN)*, pp. 4451–4458, 2017.

[18] ——, “Tensor decomposition for compressing recurrent neural network,” *International Joint Conference on Neural Networks (IJCNN)*, p. to appear, 2018.

[19] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[21] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[22] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: an ASR corpus based on public domain audio books,” in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5206–5210.

[23] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, “The Kaldi speech recognition toolkit,” in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. EPFL-CONF-192584. IEEE Signal Processing Society, 2011.