PAPER

Recurrent Neural Network Compression Based on Low-Rank Tensor Representation

Andros TJANDRA[†,††a)], ***Nonmember***, **Sakriani SAKTI**[†,††b)], ***and*** **Satoshi NAKAMURA**[†,††c)], ***Members***

SUMMARY Recurrent Neural Network (RNN) has achieved many state-of-the-art performances on various complex tasks related to the temporal and sequential data. But most of these RNNs require much computational power and a huge number of parameters for both training and inference stage. Several tensor decomposition methods are included such as CANDECOMP/PARAFAC (CP), Tucker decomposition and Tensor Train (TT) to re-parameterize the Gated Recurrent Unit (GRU) RNN. First, we evaluate all tensor-based RNNs performance on sequence modeling tasks with a various number of parameters. Based on our experiment results, TT-GRU achieved the best results in a various number of parameters compared to other decomposition methods. Later, we evaluate our proposed TT-GRU with speech recognition task. We compressed the bidirectional GRU layers inside DeepSpeech2 architecture. Based on our experiment result, our proposed TT-format GRU are able to preserve the performance while reducing the number of GRU parameters significantly compared to the uncompressed GRU.

key words: *recurrent neural network, model compression, tensor decomposition, deep learning*

1. Introduction

Modeling and predicting temporal sequential data are major task in the machine learning field. In recent years, recurrent neural network (RNN) has been a prominent choice for these tasks. Despite it has been studied for about two decades [1], [2], the renaissance just arrived recently thanks to the significant improvement of current computational power and the amount of available data. There are many state-of-the-arts in several applications such as speech recognition [3], [4] and machine translation [5]–[7] had been achieved with RNN models.

Despite the fact that RNN produced impressive performance, most RNN models are computationally expensive and have a huge number of parameters. Inside an RNN, the output are calculated by linear projection between matrices and vectors, followed by nonlinear transformations, we need multiple high-dimensional dense matrices as parameters. In-between two time-steps, we need to apply linear projection between our dense matrix with high-dimensional input and previous hidden states. Especially for state-of-the-art models on speech recognition [4] and machine translation [5], such huge models can only be implemented in high-end cluster environments because they need massive computation power and millions of parameters. These limitations restrict the creation of efficient RNN models that are fast enough for massive real-time inference or small enough to be implemented in low-end devices like mobile phones [8] or embedded systems with limited memory.

There is a trade-off between high accuracy model and huge resources requirement with fast and smaller model with low computational and memory costs. Some researchers have done notable work to minimize the accuracy loss and maximize the model efficiency. Hinton et al. [9] and Ba et al. [10] successfully compressed a large deep neural network into a smaller neural network by training the latter on the transformed softmax outputs from the former. Distilling knowledge from larger neural networks has also been successfully applied to recurrent neural network architecture by [11]. Denil et al. [12] utilized low-rank matrix decomposition of the weight matrices. A recent study by Novikov et al. [13] replaced the dense weight matrices with Tensor Train (TT) format [14] inside convolutional neural network (CNN) model. With the TT-format, they significantly compress the number of parameters and kept the model accuracy degradation to a minimum. However, to the best of our knowledge, no study has focused on compressing more complex neural networks such as RNNs with tensor-based representation.

In this paper, we utilized several tensor decomposition methods including CP-decomposition, Tucker decomposition and TT-decomposition for compressing RNN parameters*. We represent GRU RNN weight matrices with these tensor decomposition methods. First, we conduct extensive experiments on sequence modeling with a polyphonic music dataset. We compare the performances of uncompressed GRU model and three different tensor-based compressed RNN models: CP-GRU, Tucker-GRU and TT-GRU [15] on various number of parameters. From our experiment results, we conclude that TT-GRU achieved the best result in various number of parameters compared to other tensor-decomposition method. Later, we conducted another experiment on speech recognition task. We modified a popular

Manuscript received February 6, 2019.
Manuscript revised August 18, 2019.
Manuscript publicized October 17, 2019.

†The authors are with the Augmented Human Communication Lab, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan.

††The authors are with the RIKEN, Center for Advanced Intelligence Project AIP, Ikoma-shi, 630–0192 Japan.

a) E-mail: andros.tjandra.ai6@is.naist.jp
b) E-mail: ssakti@is.naist.jp
c) E-mail: s-nakamura@is.naist.jp

DOI: 10.1587/transinf.2019EDP7040

*Parts of this work were previously presented in [15]–[17]. In this paper we summarized those works and provided a more detailed description and comparison between all tensor-based RNNs.

end-to-end speech recognition model (DeepSpeech2 [4]) by replacing the GRU with TT-GRU to reduce the number of parameters. We achieve high-compression ratio and maintain the recognition accuracy from the compressed model.

In Sect. 2, we briefly review about RNN architectures and their formulations. In Sect. 3, we explain about tensor decomposition methods. In Sect. 4, we describe the details of our proposed tensor-based RNN model and how we tensorized the weight parameters inside RNN. In Sect. 5, we describe about polyphonic music modeling task, including the dataset, model settings and experimental results. In Sect. 6, we describe about speech recognition task, including the dataset, model description and experimental result. We present related works in Sect. 7. Finally, we conclude our result in Sect. 8.

2. Recurrent Neural Network

2.1 Simple Recurrent Neural Network

An RNN is a kind of neural network architecture that models sequential and temporal dependencies [18]. Typically, we define input sequence $\mathbf{x} = (x_1, \ldots, x_T)$, hidden vector sequence $\mathbf{h} = (h_1, \ldots, h_T)$ and output vector sequence $\mathbf{y} = (y_1, \ldots, y_T)$. As illustrated in Fig. 1, a simple RNN at time t is can be formulated as:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \quad (1)$$

$$y_t = g(W_{hy}h_t + b_y). \quad (2)$$

where W_{xh} represents the weight parameters between the input and hidden layer, W_{hh} represents the weight parameters between the hidden and hidden layer, W_{hy} represents the weight parameters between the hidden and output layer, and b_h and b_y represent bias vectors for the hidden and output layers. Functions $f(\cdot)$ and $g(\cdot)$ are nonlinear activation functions, such as sigmoid or tanh.

2.2 Gated Recurrent Neural Network

Simple RNNs cannot easily be used for modeling datasets with long sequences and long-term dependency because the gradient can easily vanish or explode [19], [20]. This problem is caused by the effect of bounded activation functions and their derivatives. Therefore, training a simple RNN is more complicated than training a feedforward neural network. Some researches addressed the difficulties of training simple RNNs. For example, Le et al. [21] replaced the activation function that causes the vanishing gradient with a rectifier linear (ReLU) function. With an unbounded activation function and identity weight initialization, they optimized a simple RNN for long-term dependency modeling. Martens et al. [22] used a second-order Hessian-free (HF) optimization method rather than the first-order method such as gradient descent. However, estimation of the second-order gradient requires extra computational steps. Modifying the internal structure from RNN by introducing gating mechanism also helps RNNs solve the vanishing and exploding gradient problems. The additional gating layers control the information flow from the previous states and the current input [2]. Several versions of gated RNNs have been designed to overcome the weakness of simple RNNs by introducing gating units, such as Long-Short Term Memory (LSTM) RNN and GRU RNN. In the following subsections, we explain both in more detail.

2.2.1 Long-Short Term Memory RNN

The LSTM RNN was proposed by Hochreiter et al. [2]. LSTM is a gated RNN with three gating layers and memory cells, utilizes the gating layers to control the current memory states by retaining the valuable information and forgetting the unneeded information. The memory cells store the internal information across time steps. As illustrated in Fig. 2, the LSTM hidden layer values at time t are defined by the following equations [23]:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$
$$h_t = o_t \odot \tanh(c_t)$$

where $\sigma(\cdot)$ is sigmoid activation function and i_t, f_t, o_t and c_t are respectively the input gates, the forget gates, the output gates and the memory cells. The input gates retain the

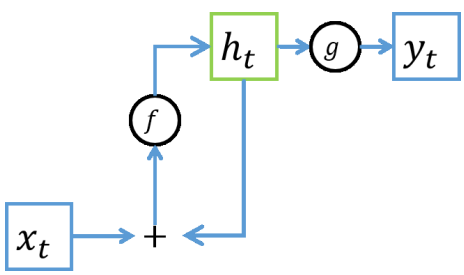

Fig. 1 Recurrent neural network

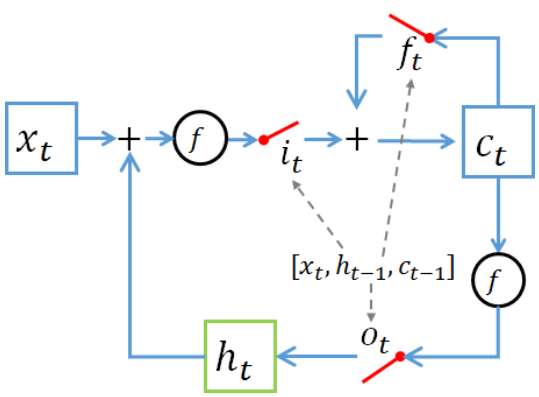

Fig. 2 Long short term memory unit.

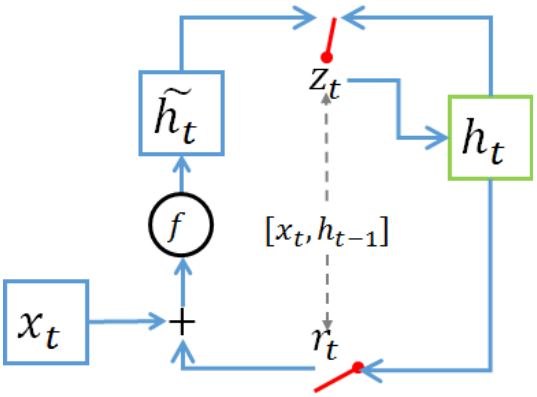

Fig. 3 Gated recurrent unit

candidate memory cell values that are useful for the current memory cell and the forget gates retain the previous memory cell values that are useful for the current memory cell. The output gates retain the memory cell values that are useful for the output and the next time-step hidden layer computation.

2.2.2 Gated Recurrent Unit RNN

The GRU RNN was proposed by Cho et al. [24] as an alternative to LSTM. There are several key differences between GRU and LSTM. First, a GRU does not have memory cells [25]. Second, instead of three gating layers, it only has two: reset gates and update gates. As illustrated in Fig. 3, the GRU hidden layer at time t is defined by the following equations [24]:

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{3}$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{4}$$
$$\tilde{h}_t = f(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{5}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{6}$$

where $\sigma(\cdot)$ is a sigmoid activation function, $f(\cdot)$ is a 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 values at time-t. The reset gates control the previous hidden layer values that are useful for the current candidate hidden layer. The update gates decide whether to keep the previous hidden layer values or replace the current hidden layer values with the candidate hidden layer values. GRU can match LSTM's performance and its convergence speed sometimes surpasses LSTM, despite having one fewer gating layer [25].

In this section, we provided the formulation and the details for several RNNs. As we can see, most of the RNNs consist of many dense matrices that represents a large number of weight parameters that are required to represent all of the RNN models. In the next section, we present an alternative RNN model that significantly reduces the number of parameters and simultaneously preserves the performance.

3. Tensor Decomposition

In this section, we explain our approaches to compress the parameters in the RNN. First, we define the tensorization process to transform the weight matrices inside the RNN model into higher order tensors. Then, we describe three tensor decompositions method called as **CANDECOMP/PARAFAC (CP) decomposition**, **Tucker decomposition** and **Tensor Train (TT) decomposition**.

3.1 Vector, Matrix and Tensor

Before we start to explain any further, we will define different notations for vectors, matrices and tensors. Vector is an one-dimensional array, matrix is a two-dimensional array and tensor is a higher-order multidimensional array. In this paper, bold lower case letters (e.g., $\mathbf{b}$) represent vectors, bold upper case letters (e.g., $\mathbf{W}$) represent matrices and bold calligraphic upper case letters (e.g., $\mathcal{W}$) represent tensors. For representing the element inside vectors, matrices and tensors, we explicitly write the index in every dimension without bold font. For example, $b(i)$ is the i-th element in vector $\mathbf{b}$, $W(p, q)$ is the element on p-th row and q-th column from matrix $\mathbf{W}$ and $\mathcal{W}(i_1, .., i_d)$ is the $i_1, .., i_d$-th index from tensor $\mathcal{W}$.

3.2 Tensor Decomposition Method

Tensor decomposition is a method for generalizing low-rank approximation from a multi-dimensional array. There are several popular tensor decomposition methods, such as Canonical polyadic (CP) decomposition, Tucker decomposition and Tensor Train decomposition. The factorization format differs across different decomposition methods. In this section, we explain briefly about CP-decomposition and Tucker decomposition.

3.2.1 CP-Decomposition

Canonical polyadic (CANDECOMP/ PARAFAC) decomposition [26]–[28] or usually referred to CP-decomposition factorizes a tensor into the sum of outer products of vectors. Assume we have a 3rd-order tensor $\mathcal{W} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$, we can approximate it with CP-decomposition:

$$\mathcal{W} \approx \sum_{r=1}^{R} \mathbf{g}_{1,r} \otimes \mathbf{g}_{2,r} \otimes \mathbf{g}_{3,r} \tag{7}$$

where $\forall r \in [1..R], \mathbf{g}_{1,r} \in \mathbb{R}^{m_1}$, $\mathbf{g}_{2,r} \in \mathbb{R}^{m_2}$, $\mathbf{g}_{3,r} \in \mathbb{R}^{m_3}$, $R \in \mathbb{Z}^+$ is the number of factors combinations (CP-rank) and $\otimes$ denotes Kronecker product operation. Elementwise, we can calculate the result by:

$$\mathcal{W}(x, y, z) \approx \sum_{r=1}^{R} g_{1,r}(x)\, g_{2,r}(y)\, g_{3,r}(z) \tag{8}$$

In Fig. 4, we provide an illustration for Eq. (7) in more details.

3.2.2 Tucker Decomposition

Tucker decomposition [28], [29] factorizes a tensor into a

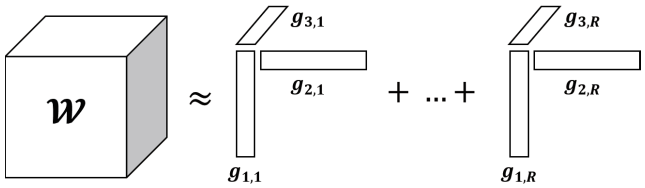

Fig. 4 CP-decomposition for 3rd-order tensor $\mathcal{W}$

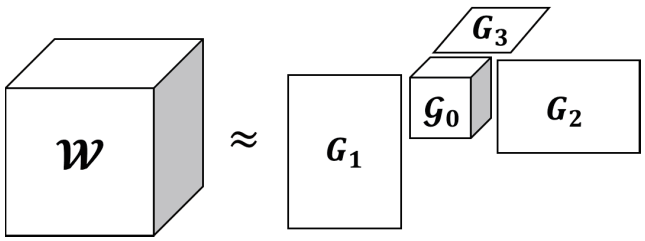

Fig. 5 Tucker decomposition for 3rd-order tensor $\mathcal{W}$

core tensor multiplied by a matrix along each mode. Assume we have a 3rd-order tensor $\mathcal{W} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$, we can approximate it with Tucker decomposition:

$$\mathcal{W} \approx \mathcal{G}_0 \times_1 \mathbf{G_1} \times_2 \mathbf{G}_2 \times_3 \mathbf{G}_3 \tag{9}$$

where $\mathcal{G}_0 \in \mathbb{R}^{r_1 \times r_2 \times r_3}$ is the core tensor, $\mathbf{G}_1 \in \mathbb{R}^{m_1 \times r_1}$, $\mathbf{G}_2 \in \mathbb{R}^{m_2 \times r_2}$, $\mathbf{G}_3 \in \mathbb{R}^{m_3 \times r_3}$ are the factor matrices and $\times_n$ is the n-th mode product operator. The mode product between a tensor $\mathcal{G}_0 \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ and a matrix $\mathbf{G}_1 \in \mathbb{R}^{m_1 \times n_1}$ is a tensor $\mathbb{R}^{m_1 \times n_2 \times n_3}$. By applying the mode products across all modes, we can recover the original $\mathcal{W}$ tensor. Elementwise, we can calculate the element from tensor $\mathcal{W}$ by:

$$\mathcal{W}(x,y,z) \approx \sum_{s_1=1}^{r_1} \sum_{s_2=1}^{r_2} \sum_{s_3=1}^{r_3} \mathcal{G}_0(s_1,s_2,s_3) \\ G_1(x,s_1)\, G_2(y,s_2)\, G_3(z,s_3) \tag{10}$$

where $x \in [1,..,m_1], y \in [1,..,m_2], z \in [1,..,m_3]$. Figure 5 gives an illustration for Eq. (9)

3.2.3 Tensor Train Decomposition

Tensor Train decomposition [14] factorizes a tensor into a collection of lower order tensors called as TT-cores. All TT-cores are connected through matrix multiplications across all tensor order to calculate the element from original tensor. Assume we have a 3rd-order tensor $\mathcal{W} \in \mathbb{R}^{m_1 \times m_2 \times m_3}$, we can approximate the element at index x, y, z by:

$$\mathcal{W}(x,y,z) \approx \sum_{s_1=1}^{r_1} \sum_{s_2=1}^{r_2} \mathcal{G}_1(x,s_1)\mathcal{G}_2(s_1,y,s_2)\mathcal{G}_3(s_2,z) \tag{11}$$

where $x \in [1,..,m_1], y \in [1,..,m_2], z \in [1,..,m_3]$ and $\mathcal{G}_1 \in \mathbb{R}^{m_1 \times r_1}$, $\mathcal{G}_2 \in \mathbb{R}^{r_1 \times m_2 \times r_2}$, $\mathcal{G}_3 \in \mathbb{R}^{r_2 \times m_3}$ as the TT-cores. Figure 6 gives an illustration for Eq. (11).

To generalized TT-decomposition for d-dimensional tensor, TT-decomposition equation can be described as:

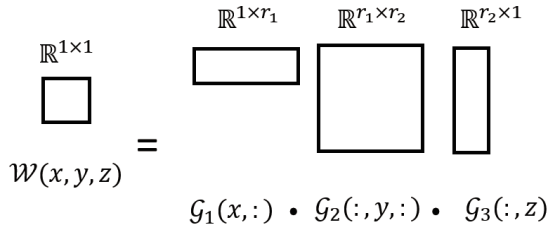

Fig. 6 Tensor Train decomposition for 3rd-order tensor $\mathcal{W}$

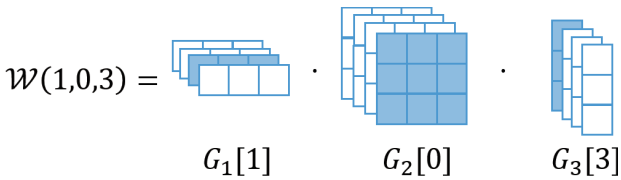

Fig. 7 Representing a tensor $\mathcal{W}$ element at $(1,0,3)$ using 3 TT-cores $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{G}_3$. Blue shaded vectors or matrices are used for chain multiplication

$$\mathcal{W}(j_1,j_2,..,j_{d-1},j_d) = \\ \mathcal{G}_1(j_1) \cdot \mathcal{G}_2(:,j_2,:) \cdot \ldots \cdot \mathcal{G}_{d-1}(:,j_{d-1},:) \cdot \mathcal{G}_d(:,j_d). \tag{12}$$

where $j_k \in [1,..,m_k]$. For all vectors or matrices sliced from tensor $\mathcal{G}_k$, if $\mathcal{G}_k(:,j_k,:)$ 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 we call them TT-rank from tensor $\mathcal{W}$. The set of matrices $\mathcal{G}_k = \{\mathcal{G}_k(:,j_k,:)\}_{j_k=1}^{m_k}$ where the matrices are spanned in the same index are called TT-core.

We can generalized the TT-format equation in detail by enumerating the index $q_{k-1} \in \{1,..,r_{k-1}\}$ and $q_k \in \{1,..,r_k\}$ in matrix $\mathcal{G}_k(j_k)$ across all $k \in \{1,..,d\}$:

$$\mathcal{W}(j_1,j_2,..,j_{d-1},j_d) = \\ \sum_{q_0,..,q_d} \mathcal{G}_1(q_0,j_1,q_1) \ \ldots \ \mathcal{G}_d(q_{d-1},j_d,q_d). \tag{13}$$

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

Lastly, to create an intuitive example on how to represent tensor using set of TT-cores, we illustrate how to represent a tensor $\mathcal{W}$ element at $(1,0,3)$ with 3 TT-cores in Fig. 7.

4. Proposed Tensor-Based RNN

In this section, we describe our proposed approach to compress RNN model using various tensor decomposition method that we have described above. First, we will explain about linear layer tensorization. After that, we rewrite the

RNN equation based on the from the matrices weight format into the tensorized weight format.

4.1 Representing Linear Transformation on Tensorized Weight

Most of RNN equations are composed by multiplication between the input vector and their corresponding weight matrix:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{14}$$

where $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the weight matrix, $\mathbf{b} \in \mathbb{R}^{M}$ is the bias vector and $\mathbf{x} \in \mathbb{R}^{N}$ is the input vector. Thus, most of RNN parameters are used to represent the weight matrices. To reduce the number of parameters significantly, we need to represent the weight matrices with the factorization of higher-order tensor. First, we apply tensorization on the weight matrices. Tensorization is the process to transform a lower-order dimensional array into a higher-order dimensional array. In our case, we tensorize RNN weight matrices into tensors. Given a weight matrix $\mathbf{W} \in \mathbb{R}^{M \times N}$, we can represent them as a tensor $\mathcal{W} \in \mathbb{R}^{m_1 \times m_2 \times .. \times m_d \times n_1 \times n_2 \times .. \times n_d}$ where $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$. For mapping each element in matrix $\mathbf{W}$ to tensor $\mathcal{W}$, we define one-to-one mapping between row-column and tensor index with bijective functions $\mathbf{f}_i : \mathbb{Z}_+ \rightarrow \mathbb{Z}_+^d$ and $\mathbf{f}_j : \mathbb{Z}_+ \rightarrow \mathbb{Z}_+^d$. Function $\mathbf{f}_i$ transforms each row $p \in \{1, .., M\}$ into $\mathbf{f}_i(p) = [i_1(p), .., i_d(p)]$ and $\mathbf{f}_j$ transforms each column $q \in \{1, .., N\}$ into $\mathbf{f}_j(q) = [j_1(q), .., j_d(q)]$. Following this, we can access the value from matrix $W(p, q)$ in the tensor $\mathcal{W}$ with the index vectors generated by $\mathbf{f}_i(p)$ and $\mathbf{f}_j(q)$ with these bijective functions.

After we determine the shape of the weight tensor, we choose one of the tensor decomposition methods (e.g., CP-decomposition (Sect. 3.2.1), Tucker decomposition (Sect. 3.2.2) or Tensor Train (Sect. 3.2.3)) to represent and reduce the number of parameters from the tensor $\mathcal{W}$. In order to represent matrix-vector products inside RNN equations, we need to reshape the input vector $\mathbf{x} \in \mathbb{R}^N$ into a tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times .. \times n_d}$ and the bias vector $\mathbf{b} \in \mathbb{R}^M$ into a tensor $\mathcal{B} \in \mathbb{R}^{m_1 \times .. \times m_d}$. Therefore, we can reformulate the Eq. (14) to calculate $y(p)$ elementwise with:

$$\mathcal{Y}(\mathbf{f}_i(p)) = \sum_{j_1, .., j_d} \mathcal{W}(\mathbf{f}_i(p), j_1, .., j_d)\ \mathcal{X}(j_1, .., j_d) + \mathcal{B}(\mathbf{f}_i(p)) \tag{15}$$

by enumerating all columns q position with $j_1, .., j_d$ and $\mathbf{f}_i(p) = [i_1(p), .., i_d(p)]$.

Based on the Eq. (15), we rewrite the linear layer equation differently depends on the the decomposition method:

1. **CP-decomposition**:
   For CP-decomposition, we represent our tensor $\mathcal{W}$ with multiple factors $\mathbf{gm}_{k,r}, \mathbf{gn}_{k,r}$ where $\forall k \in [1..d]\, \forall r \in [1..R]$, $(\mathbf{gm}_{k,r} \in \mathbb{R}^{m_k}, \mathbf{gn}_{k,r} \in \mathbb{R}^{n_k})$. From here, we replace Eq. (15) with:

$$\begin{aligned}\mathcal{Y}(\mathbf{f}_i(p)) &= \sum_{j_1, .., j_d} \left(\sum_{r=1}^{R} \prod_{k=1}^{d} gm_{k,r}(i_k(p)) gn_{k,r}(j_k) \right) \mathcal{X}(j_1, .., j_d) \\ &\quad + \mathcal{B}(\mathbf{f}_i(p)).\end{aligned} \tag{16}$$

   By using CP-decomposition for representing the weight matrix $\mathbf{W}$, we reduce the number of parameters from $M \times N$ into $R * (\sum_{k=1}^{d} m_k + n_k)$.

2. **Tucker decomposition**:
   For Tucker decomposition, we represent our tensor $\mathcal{W}$ with a tensor core $\mathcal{G}_0 \in \mathbb{R}^{r_1 \times .. \times r_d \times r_{d+1} \times .. \times r_{2d}}$ where $\forall k \in [1..d]$, $r_k < m_k$ and $\forall k \in [1..d]$, $r_{d+k} < n_k$ and multiple factor matrices $\mathbf{GM}_k, \mathbf{GN}_k$, where $\forall k \in [1..d]$, $(\mathbf{GM}_k \in \mathbb{R}^{m_k \times r_k}, \mathbf{GN}_k \in \mathbb{R}^{n_k \times r_{d+k}})$. Generally, the tensor core ranks $r_1, r_2, .., r_d$ are corresponding to the row in tensor index and $r_{d+1}, r_{d+2}, .., r_{2d}$ are corresponding to the column in tensor index. From here, we replace Eq. (15) with:

$$\begin{aligned}\mathcal{Y}(\mathbf{f}_i(p)) = \sum_{j_1, .., j_d} \Bigg(\sum_{s_1, .. s_d, s_{d+1}, .., s_{2d}}^{r_1, .., r_d, r_{d+1}, .., r_{2d}} \mathcal{G}_0(s_1, .. s_d, s_{d+1}, .., s_{2d}) \\ \prod_{k=1}^{d} GM_k(i_k(p), s_k) GN_k(j_k, s_{d+k}) \Bigg) \mathcal{X}(j_1, .., j_d) \\ + \mathcal{B}(\mathbf{f}_i(p)).\end{aligned} \tag{17}$$

   By using Tucker decomposition for representing the weight matrix $\mathbf{W}$, we reduce the number of parameters from $M \times N$ into $\sum_{k=1}^{d} (m_k * r_k + n_k * r_{d+k}) + (\prod_{k=1}^{2d} r_k)$.

3. **Tensor-train decomposition**:
   For Tensor-train decomposition, we represent our tensor $\mathcal{W}$ with multiple TT-cores $\mathcal{G}_{m_1..m_d}$ and $\mathcal{G}_{n_1..n_d}$. TT-cores $\mathcal{G}_k \in \mathbb{R}^{r_{k-1} \times m_k \times r_k}$, $\forall k \in [1..d]$ are corresponding to the row in tensor index and TT-cores $\mathcal{G}_{d+k} \in \mathbb{R}^{r_{d+k-1} \times n_k \times r_{d+k}}$, $\forall k \in [1..d]$ are corresponding to the column in tensor index. $r_{k-1}..r_d$ are the TT-ranks for row TT-cores and $r_{d+k-1}..r_{2d}$ are the TT-ranks for column TT-cores. From here, we replace Eq. (15) with:

$$\begin{aligned}\mathcal{Y}(\mathbf{f}_i(p)) = \sum_{j_1, .., j_d} &(\mathcal{G}_1(:, i_1(p), :)..\mathcal{G}_d(:, i_d(p), :) \\ &\cdot \mathcal{G}_{d+1}(:, j_1, :)..\mathcal{G}_{2d}(:, j_d, :))\, \mathcal{X}(j_1, .., j_d) \\ &+ \mathcal{B}(\mathbf{f}_i(p)).\end{aligned} \tag{18}$$

   By using TT-decomposition for representing the weight matrix $\mathcal{W}$, we reduce the number of parameters from $M \times N$ into $\sum_{k=1}^{d} ((r_{k-1} * m_k * r_k) + (r_{d+k-1} * n_k * r_{2d}))$. We can control the shape of TT-cores $\{\mathcal{G}_k\}_{i=1}^{2d}$ by choosing factor M as $\{m_k\}_{k=1}^{d}$ and N as $\{n_k\}_{k=1}^{d}$ as long as the number of factors is equal between M and N. We can also define TT-rank and treat them as a hyper-parameter. In general, if we use a smaller TT-rank, we will get more efficient models but this action restricts our model to learn more complex representation. If we use a larger TT-rank, we get more flexibility to

express our weight parameters but we sacrifice model efficiency.

4.2 Compressing Simple RNN

We represent a simple RNN in tensor-based format. From Sect. 2.1, we focus our attention on two dense weight matrices: (W_{xh}, W_{hh}). Previously, we defined $\mathbf{W}_{xh} \in \mathbb{R}^{M \times N}$ as input-to-hidden parameters and $\mathbf{W}_{hh} \in \mathbb{R}^{M \times M}$ as hidden-to-hidden parameters and b_h .

First, we factorize matrix shape M into $\prod_{k=1}^{d} m_k$ and N into $\prod_{k=1}^{d} n_k$. Next, we determine which tensor decomposition method to represent the weight matrices. We will substitute weight matrix $\mathbf{W}_{xh}$ with tensor $\boldsymbol{\mathcal{W}}_{xh}$ and $\mathbf{W}_{hh}$ with tensor $\boldsymbol{\mathcal{W}}_{hh}$. We define bijective functions $\mathbf{f}_i^x$ and $\mathbf{f}_i^h$ to access row $p \in [1..M]$ from W_{xh} and W_{hh} in the tensor-based representation. We rewrite our simple RNN formulation to calculate h_t in Eq. (1):

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

where $\boldsymbol{\mathcal{X}}$ is the tensor representation of input x_t and $\boldsymbol{\mathcal{H}}_{t-1}$ is the tensor representation of previous hidden states h_{t-1}.

4.3 Compressing GRU RNN

In this section, we apply tensor-based format to represent a gated RNN. Among several RNN architectures with gating mechanism, we choose GRU to be reformulated in tensor-based representation because it has less complex formulation and similar performance as LSTM. For the rest of this paper, we called Tucker-GRU if we used Tucker decomposition, CP-GRU if we used CP-decomposition and TT-GRU if we used TT-decomposition to replace the weight matrices inside GRU. In Sect. 2.2.2, we focus on the following six dense weight matrices: $(W_{xr}, W_{hr}, W_{xz}, W_{hz}, W_{xh}$, and $W_{hh})$. 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, the update gate, the candidate hidden layer, and $W_{hr}, W_{hz}, W_{hh} \in \mathbb{R}^{M \times M}$ are respectively parameters for projecting previous hidden layer into the reset gate, the update gate and candidate hidden layer.

We factorize matrix shape M into $\prod_{k=1}^{d} m_k$ and N into $\prod_{k=1}^{d} n_k$. All weight matrices $(W_{xr}, W_{hr}, W_{xz}, W_{hz}, W_{xh}, W_{hh})$ are substituted with tensors $(\boldsymbol{\mathcal{W}}_{xr}, \boldsymbol{\mathcal{W}}_{hr}, \boldsymbol{\mathcal{W}}_{xz}, \boldsymbol{\mathcal{W}}_{hz}, \boldsymbol{\mathcal{W}}_{xh}, \boldsymbol{\mathcal{W}}_{hh})$ in tensor-based weight representation. 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 tensor-based weight representation. We rewrite the GRU formulation to calculate r_t in Eq. (3):

$$
\begin{aligned}
a_t^{xr}(p) &= \sum_{j_1,..,j_d} \mathcal{W}_{xr}(\mathbf{f}_i^x(p), [j_1, .., j_d]) \cdot \mathcal{X}_t\,(j_1, .., j_d) \\
a_t^{hr}(p) &= \sum_{j_1,..,j_d} \mathcal{W}_{hr}(\mathbf{f}_i^h(p), [j_1, .., j_d]) \cdot \mathcal{H}_{t-1}\,(j_1, .., j_d) \\
a_t^{xr} &= [a_t^{xr}(1), .., a_t^{xr}(M)] \\
a_t^{hr} &= \left[a_t^{hr}(1), .., a_t^{hr}(M)\right] \\
r_t &= \sigma(a_t^{xr} + a_t^{hr} + b_r).
\end{aligned} \tag{19}
$$

Next, we rewrite the GRU formulation to calculate z_t in Eq. (4):

$$
\begin{aligned}
a_t^{xz}(p) &= \sum_{j_1,..,j_d} \mathcal{W}_{xz}(\mathbf{f}_i^x(p), [j_1, .., j_d]) \cdot \mathcal{X}_t\,(j_1, .., j_d) \\
a_t^{hz}(p) &= \sum_{j_1,..,j_d} \mathcal{W}_{hz}(\mathbf{f}_i^h(p), [j_1, .., j_d]) \cdot \mathcal{H}_{t-1}\,(j_1, .., j_d) \\
a_t^{xz} &= [a_t^{xz}(1), .., a_t^{xz}(M)] \\
a_t^{hz} &= \left[a_t^{hz}(1), .., a_t^{hz}(M)\right] \\
z_t &= \sigma(a_t^{xz} + a_t^{hz} + b_z).
\end{aligned} \tag{20}
$$

Finally, we rewrite the GRU formulation to calculate $\tilde{h}_t$ in Eq. (5):

$$
\begin{aligned}
a_t^{xh}(p) &= \sum_{j_1,..,j_d} \mathcal{W}_{xh}(\mathbf{f}_i^x(p), [j_1, .., j_d]) \cdot \mathcal{X}_t\,(j_1, .., j_d) \\
a_t^{hh}(p) &= \sum_{j_1,..,j_d} \mathcal{W}_{hh}(\mathbf{f}_i^h(p), [j_1, .., j_d]) \cdot \\
&\qquad (\mathcal{R}_t\,(j_1, .., j_d) \cdot \mathcal{H}_{t-1}\,(j_1, .., j_d)) \\
a_t^{xh} &= \left[a_t^{xh}(1), .., a_t^{xh}(M)\right] \\
a_t^{hh} &= \left[a_t^{hh}(1), .., a_t^{hh}(M)\right] \\
\tilde{h}_t &= f(a_t^{xh} + a_t^{hh} + b_h).
\end{aligned} \tag{21}
$$

After r_t, z_t and $\tilde{h}_t$ are calculated, we calculate h_t on Eq. (6) with standard operations like element-wise sum and multiplication.

In practice, we could assign a different d for each weight tensor as long as the input data dimension can also be factorized into the d values. The choice of tensor decomposition method, ranks, and factors shape are determined by the user and treated as hyperparameter. However, to simplify our implementation we use the same d for both the input and hidden factorization size. For TT-GRU, we also use the same factorizations $M = \prod_{k=1}^{d} m_k$ and $N = \prod_{k=1}^{d} n_k$ for all weight tensors.

We do not substitute 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 the matrix multiplication between a weight matrix and the input layer or the previous hidden layer.

4.4 Tensor Core and Factors Initialization Trick

Because of the large number of recursive matrix

multiplications, followed by some nonlinearity (e.g, sigmoid, tanh), the gradient from the hidden layer will diminish after several time-step [30]. Consequently, training recurrent neural networks is much harder compared to standard feedforward neural networks.

Even worse, we decompose the weight matrix into multiple smaller tensors or matrices, thus the number of multiplications needed for each calculation increases multiple times. Therefore, we need a better initialization trick on the tensor cores and factors to help our model convergences in the early training stage.

In this work, we follow Glorot et al. [31] by initializing the weight matrix with a certain variance. We assume that our original weight matrix $\mathbf{W}$ has a mean 0 and the variance σ_W^2. We utilize the basic properties from a sum and a product variance between two independent random variables.

Definition 4.1. Let X and Y be independent random variables with the mean 0, then the variance from the sum of X and Y is $Var(X + Y) = Var(X) + Var(Y)$

Definition 4.2. Let X and Y be independent random variables with the mean 0, then the variance from the product of X and Y is $Var(X * Y) = Var(X) * Var(Y)$

After we decided the target variance σ_w^2 for our original weight matrix, now we need to derive the proper initialization rules for the tensor core and factors. We calculate the variance for tensor core and factors by observing the number of sum and product operations and utilize the variance properties from Def. 4.1 and 4.2. Here, we listed different initialization strategies for each tensor decomposition methods:

1. **CP-decomposition**:
   For weight tensor $\mathcal{W}$ based on the CP-decomposition, we can calculate σ_g as the standard deviation for all factors $\mathbf{gm}_{k,r}, \mathbf{gn}_{k,r}$ with:

$$\sigma_g = \sqrt[4d]{\frac{\sigma_w^2}{R}} \tag{22}$$

   and initialize $\mathbf{gm}_{k,r}, \mathbf{gn}_{k,r} \sim \mathcal{N}(0, \sigma_g^2)$.

   Proof. Define 1) a dense matrix $\mathbf{W}$ and each element are initialized with independent random variable w, 2) all tensor factors $\mathbf{gm}, \mathbf{gn}$ element are initialized with independent random variable g. We approximate variable w by series of summation and multiplication from elements of tensor factors random variable g (Eq. (16)).

$$\mathbf{W}(p, q) = \sum_{r=1}^{R} \prod_{k=1}^{d} \mathbf{gm}_{k,r}(i_k(p))\mathbf{gn}_{k,r}(j_k(q)) \tag{23}$$

   We replace $\mathbf{W}(p, q)$ with random variable w and $\mathbf{gm}_{k,r}(i_k(p)), \mathbf{gn}_{k,r}(j_k(q))$ with random variable g.

$$w = \sum_{r=1}^{R} \prod_{k=1}^{d} g * g \tag{24}$$

$$w = \sum_{r=1}^{R} g^{2d} \tag{25}$$

$$w = Rg^{2d} \tag{26}$$

   Apply variance for both sides.

$$Var(w) = Var\left(Rg^{2d}\right) \tag{27}$$

$$Var(w) = RVar\left(g^{2d}\right) \tag{28}$$

$$Var(w) = RVar\,(g)^{2d} \tag{29}$$

$$\sigma_w^2 = Var(w) \tag{30}$$

$$\sigma_g^2 = Var(g) \tag{31}$$

$$\frac{\sigma_w^2}{R} = (\sigma_g^2)^{2d} \tag{32}$$

$$\sqrt[2d]{\frac{\sigma_w^2}{R}} = \sigma_g^2 \tag{33}$$

$$\sqrt[4d]{\frac{\sigma_w^2}{R}} = \sigma_g \tag{34}$$

   $*$ **Notes**: g between different factors and positions are not a same random variable (we just write as g for sake of simplicity). Therefore, for the case where multiple sum of g, we use Definition 4.1, **not** of $Var(aX) = a^2Var(X)$. For the case of multiple multiplication of g, we use Definition 4.2, **not** based on Chi-Square random variable and their variance calculation. □

2. **Tucker decomposition**:
   For weight tensor $\mathcal{W}$ based on the Tucker decomposition, we can calculate σ_g as the standard deviation for the core tensor $\mathcal{G}_0$ and the factor matrices $\mathbf{GM}_k, \mathbf{GN}_k$ with:

$$\sigma_g = \sqrt[(4d+2)]{\frac{\sigma_w^2}{\prod_{k=1}^{2d} r_k}} \tag{35}$$

   and initialize $\mathcal{G}_0, \mathbf{GM}_k, \mathbf{GN}_k \sim \mathcal{N}(0, \sigma_g^2)$.

   Proof. Define 1) a dense matrix $\mathbf{W}$ and each element are initialized with independent random variable w, 2) all tensor factors $\mathcal{G}_\mathbf{0}, \mathbf{GM}, \mathbf{GN}$ element are initialized with independent random variable g. We approximate variable w by series of summation and multiplication from elements of tensor factors random variable g (Eq. (17)).

$$\mathbf{W}(p, q) = \sum_{s_1,..,s_d,s_{d+1},..,s_{2d}}^{r_1,..,r_d,r_{d+1},..,r_{2d}} \mathcal{G}_0(s_1, ..s_d, s_{d+1}, .., s_{2d}) \prod_{k=1}^{d} GM_k(i_k(p), s_k)GN_k(j_k(q), s_{d+k}) \tag{36}$$

   We replace $\mathbf{W}(p, q)$ with random variable w and $\mathcal{G}_\mathbf{0}, \mathbf{GM}, \mathbf{GN}$ with random variable g.

$$w = \sum_{s_1,..,s_d,s_{d+1},..,s_{2d}}^{r_1,..,r_d,r_{d+1},..,r_{2d}} g \prod_{k=1}^{d} g * g \tag{37}$$

$$w = \sum_{s_1,..,s_d,s_{d+1},..,s_{2d}}^{r_1,..,r_d,r_{d+1},..,r_{2d}} g * g^{2d} \tag{38}$$

$$w = \sum_{s_1,..,s_d,s_{d+1},..,s_{2d}}^{r_1,..,r_d,r_{d+1},..,r_{2d}} g^{2d+1} \tag{39}$$

$$w = \prod_{k=1}^{2d} r_k g^{2d+1} \tag{40}$$

Apply variance for both sides.

$$Var(w) = Var\left(\prod_{k=1}^{2d} r_k g^{2d+1}\right) \tag{41}$$

$$Var(w) = \prod_{k=1}^{2d} r_k Var\left(g^{2d+1}\right) \tag{42}$$

$$Var(w) = \prod_{k=1}^{2d} r_k Var(g)^{2d+1} \tag{43}$$

$$\sigma_w^2 = Var(w) \tag{44}$$

$$\sigma_g^2 = Var(g) \tag{45}$$

$$\frac{\sigma_w^2}{\prod_{k=1}^{2d} r_k} = (\sigma_g^2)^{2d+1} \tag{46}$$

$$\sqrt[2d+1]{\frac{\sigma_w^2}{\prod_{k=1}^{2d} r_k}} = \sigma_g^2 \tag{47}$$

$$\sqrt[4d+2]{\frac{\sigma_w^2}{\prod_{k=1}^{2d} r_k}} = \sigma_g \tag{48}$$

□

3. **Tensor-train decomposition**:
For weight tensor $\mathcal{W}$ based on the Tensor Train decomposition, we can calculate σ_g as the standard deviation for all the TT-cores $\mathcal{G}_k$ with:

$$\sigma_g = \sqrt[(4d-2)]{\frac{\sigma_w^2}{\prod_{k=0}^{2d} r_k}} \tag{49}$$

and initialize $\mathcal{G}_k \sim \mathcal{N}(0, \sigma_g^2)$.

Proof. Define 1) a dense matrix $\mathbf{W}$ and each element are initialized with independent random variable w, 2) all tensor cores $\mathcal{G}_{\mathbf{m_1},..,\mathbf{m_d}}, \mathcal{G}_{\mathbf{n_1},..,\mathbf{n_d}}$ element are initialized with independent random variable g. We approximate variable w by series of summation and multiplication from elements of tensor factors random variable g (Eq. (18)).

$$\begin{aligned}\mathbf{W}(p,q) = \mathcal{G}_1(:, i_1(p), :)\mathcal{G}_d(:, i_d(p), :) \\ \mathcal{G}_{d+1}(:, j_1(q), :)\mathcal{G}_{2d}(:, j_d(q), :)\end{aligned} \tag{50}$$

We replace $\mathbf{W}(p,q)$ with random variable w and $\mathcal{G}_{\mathbf{m_1},..,\mathbf{m_d}}, \mathcal{G}_{\mathbf{n_1},..,\mathbf{n_d}}$ with random variable g.

$$w = \prod_{k=0}^{2d} r_k g^{(2d-1)} \tag{51}$$

Apply variance for both sides.

$$Var(w) = Var\left(\prod_{k=0}^{2d} r_k g^{(2d-1)}\right) \tag{52}$$

$$Var(w) = \prod_{k=0}^{2d} r_k Var\left(g^{(2d-1)}\right) \tag{53}$$

$$Var(w) = \prod_{k=0}^{2d} r_k Var\,(g)^{(2d-1)} \tag{54}$$

$$\sigma_w^2 = Var(w) \tag{55}$$

$$\sigma_g^2 = Var(g) \tag{56}$$

$$\sigma_w^2 = \prod_{k=0}^{2d} r_k \sigma_g^{2(2d-1)} \tag{57}$$

$$\sigma_w^2 = \prod_{k=0}^{2d} r_k \sigma_g^{4d-2} \tag{58}$$

$$\frac{\sigma_w^2}{\prod_{k=0}^{2d} r_k} = \sigma_g^{4d-2} \tag{59}$$

$$\sqrt[4d-2]{\frac{\sigma_w^2}{\prod_{k=0}^{2d} r_k}} = \sigma_g \tag{60}$$

□

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

5. Experiments 1: Polyphonic Music Modeling

In this section, we describe our dataset and all model configurations. We performed experiments with three different tensor-decompositions (CP decomposition, Tucker decomposition and TT decomposition) to compress our GRU and also the baseline GRU. In the end, we report our experiment results and compare various tensor decomposition method with different settings and number of parameters. Our codes are available at `https://github.com/androstj/tensor_rnn`.

5.1 Dataset

We evaluated our models with sequential modeling tasks. We used a polyphonic music dataset [32] which contains 4 different datasets[†]: Nottingham, MuseData, PianoMidi and JSB Chorales. For each active note in all time-step, we set the value as 1, otherwise 0. Each dataset consists of at least 7 hours of polyphonic music and the total is ± 67 hours. In Fig. 8, we visualize the dataset and the task for this experiment.

[†]Dataset are downloaded from: http://www-etud.iro.umontreal.ca/~boulanni/icml2012

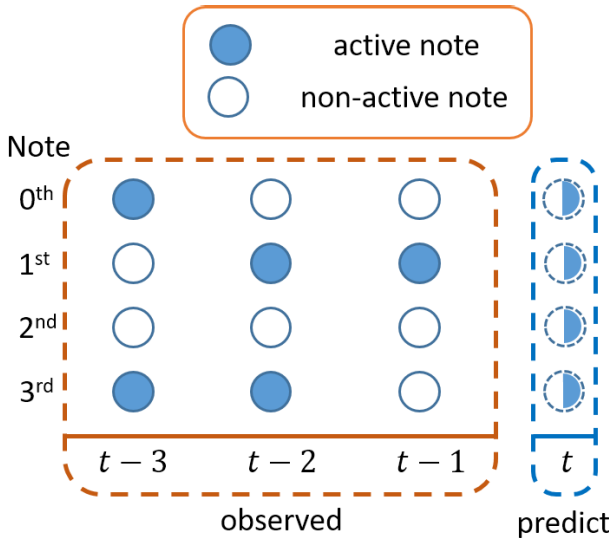

Fig. 8 A simple illustration for polyphonic music dataset. For each time-step, we have a vector of binary number. The colored circle denotes the active note and the blank circle denotes the non-active note. The main objective in this task is to predict which notes will be active at time-step t given the information from previous active notes at time-step $t-1, t-2, \ldots$, etc.

5.2 Models

We evaluate several models in this paper: GRU-RNN (no compression), CP-GRU (weight compression via CP decomposition), Tucker-GRU (weight compression via Tucker decomposition), TT-GRU [15] (compressed weight with TT-decomposition). For each timestep, the input and output targets are vectors of 88 binary value. The input vector is projected by a linear layer with 256 hidden units, followed by LeakyReLU [33] activation function. For the RNN model configurations, we enumerate all the details in the following list:

1. GRU
   - Input size (N): 256
   - Hidden size (M): 512
2. Tensor-based GRU
   - Input size (N): 256
   - Tensor input shape ($n_{1..4}$): $4 \times 4 \times 4 \times 4$
   - Hidden size (M): 512
   - Tensor hidden shape ($m_{1..4}$): $8 \times 4 \times 4 \times 4$

   a. CP-GRU
      - CP-Rank (R): $[10, 30, 50, 80, 110]$

   b. Tucker-GRU
      - Core ($\mathcal{G}_0$) shape:
        - $(2 \times 2 \times 2 \times 2) \times (2 \times 2 \times 2 \times 2)$
        - $(2 \times 3 \times 2 \times 3) \times (2 \times 3 \times 2 \times 3)$
        - $(2 \times 3 \times 2 \times 4) \times (2 \times 3 \times 2 \times 4)$
        - $(2 \times 4 \times 2 \times 4) \times (2 \times 4 \times 2 \times 4)$
        - $(2 \times 3 \times 3 \times 4) \times (2 \times 3 \times 3 \times 4)$

   c. TT-GRU
      - TT-ranks:
        - $(1 \times 3 \times 3 \times 3 \times 1)$
        - $(1 \times 5 \times 5 \times 5 \times 1)$
        - $(1 \times 7 \times 7 \times 7 \times 1)$
        - $(1 \times 9 \times 9 \times 9 \times 1)$
        - $(1 \times 9 \times 9 \times 9 \times 1)$

In this task, the training criterion is to minimize the negative log-likelihood (NLL). In evaluation, we measured two different scores: NLL and accuracy (ACC). For calculating the accuracy, we follow Bay et al. [34] formulation:

$$ACC = \frac{\sum_{t=1}^{T} TP(t)}{\sum_{t=1}^{T} (TP(t) + FP(t) + FN(t))} \tag{61}$$

where $TP(t)$, $FP(t)$, $FN(t)$ is the true positive, false positive and false negative at time-t. We only used true positive (TP), false positive (FP), false negative (FN) and ignored the true negative (TN) because most of the notes were turned off or zero in the dataset.

For training models, we use Adam [35] algorithm for our optimizer. To stabilize our training process, we clip our gradient when the norm $\|\nabla w\| > 5$. For fair comparisons, we performed a grid search over learning rates $(1e-2, 5e-3, 1e-3)$ and dropout probabilities $(0.2, 0.3, 0.4, 0.5)$. The best model based on loss in validation set will be used for the test set evaluation.

5.3 Result and Discussion

We report results of our experiments in Table 1. For the baseline model, we choose standard GRU-RNN without any compression on the weight matrices. For the comparison between compressed models (CP-GRU, Tucker-GRU and TT-GRU), we run each model with 5 different configurations and varied the number of parameters ranged from 2232 up to 12184. In Figs. 9–12, we plot the negative log-likelihood (NLL) score corresponding to the number of parameters for each model. From our results, we observe that TT-GRU performed better than Tucker-GRU in every experiments with similar number of parameters. In some datasets (e.g., PianoMidi, MuseData, Nottingham), CP-GRU has better results compared to Tucker-GRU and achieves similar performance (albeit slightly worse) as TT-GRU when the number of parameters are greater than 6000. Overall, TT-GRU performed the best in most of the settings. Therefore, for the next experiment, we will apply TT-GRU as our main compression method.

6. Experiment II: Speech Recognition

In this task, we will evaluate our proposed TT-GRU model

Table 1 Comparison between all models and their configurations based on the number of parameters, negative log-likelihood and accuracy of polyphonic test set

Model	Config	Param	Dataset							
			Nottingham		JSB		PianoMidi		MuseData	
			NLL	ACC	NLL	ACC	NLL	ACC	NLL	ACC
GRU IN:256 OUT:512		1181184	3.369	71.1	8.32	30.24	7.53	27.19	7.12	36.30
CP-GRU IN: 4,4,4,4 OUT: 8,4,4,4	**Rank**									
	10	2456	3.79	67.51	8.60	27.29	8.15	19.03	7.87	27.32
	30	4296	3.48	69.85	8.49	28.33	7.68	25.03	7.27	36.19
	50	6136	3.46	69.56	8.40	28.47	7.66	26.18	7.23	36.34
	80	8896	3.43	69.73	8.41	27.88	7.61	28.28	7.19	36.57
	110	11656	3.34	70.42	8.41	29.45	7.60	27.36	7.18	36.89
TUCKER-GRU IN: 4,4,4,4 OUT: 8,4,4,4	**Cores**									
	2,2,2,2	2232	3.71	68.30	8.57	27.28	7.98	20.79	7.81	29.94
	2,3,2,3	4360	3.64	68.63	8.48	28.10	7.75	24.92	7.38	34.20
	2,3,2,4	6408	3.55	69.10	8.44	28.06	7.73	25.66	7.69	32.50
	2,4,2,4	10008	3.52	69.18	8.41	27.70	7.75	24.46	7.38	35.58
	2,3,3,4	12184	3.41	70.23	8.43	29.03	7.69	25.26	7.43	33.63
TT-GRU IN: 4,4,4,4 OUT: 8,4,4,4	**TT-rank**									
	1,3,3,3,1	2688	3.49	69.49	8.37	28.41	7.60	26.95	7.49	34.99
	1,5,5,5,1	4096	3.45	69.81	8.38	28.86	7.58	27.46	7.50	33.37
	1,7,7,7,1	6016	3.40	70.72	8.37	28.83	7.57	27.58	7.23	36.53
	1,9,9,9,1	8448	3.35	70.82	8.36	29.32	7.58	27.62	7.20	37.81
	1,11,11,11,1	11392	3.38	70.51	8.37	29.55	7.58	28.07	7.16	36.54

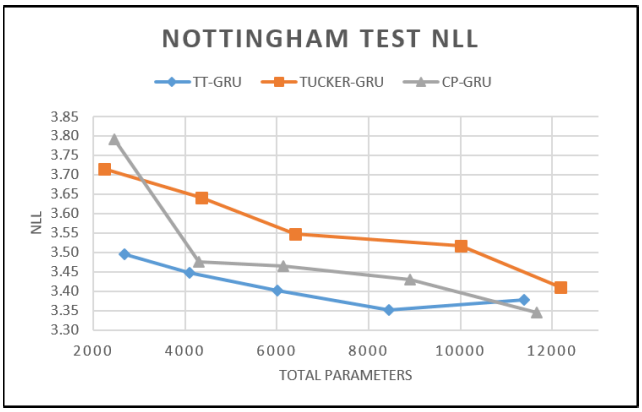

Fig. 9 NLL comparison between TT-GRU, Tucker-GRU, and CP-GRU on Nottingham test set

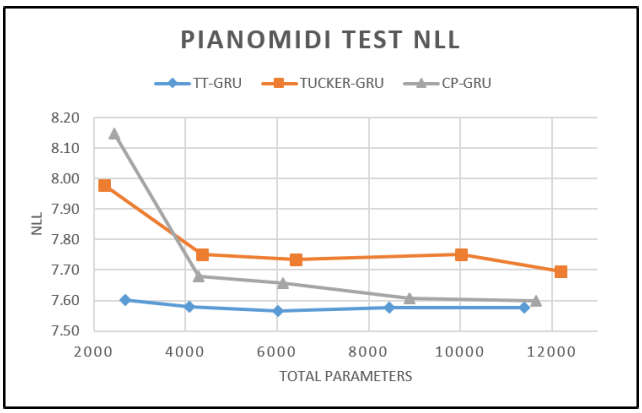

Fig. 11 NLL comparison between TT-GRU, Tucker-GRU, and CP-GRU on PianoMidi test set

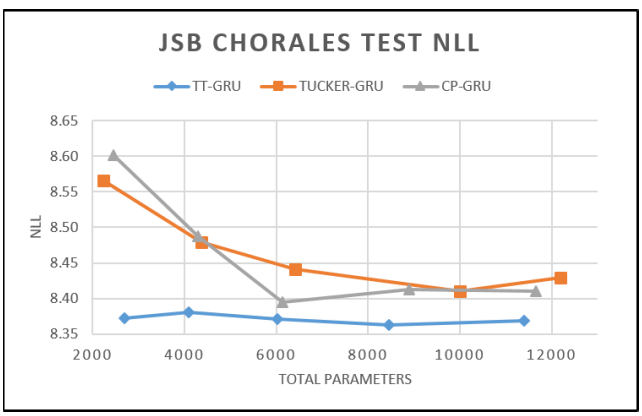

Fig. 10 NLL comparison between TT-GRU, Tucker-GRU, and CP-GRU on JSB Chorales test set

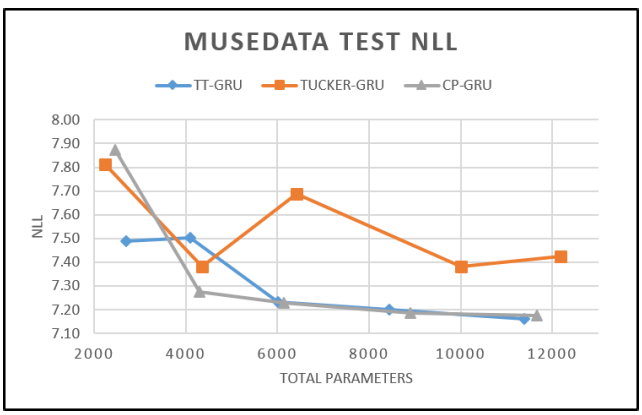

Fig. 12 NLL comparison between TT-GRU, Tucker-GRU, and CP-GRU on MuseData test set

on speech recognition task. We describe our dataset for speech recognition task and our end-to-end ASR model for this task. In the end, we report our experiment results by comparing the performance of uncompressed model with TT-GRU compressed model.

6.1 Dataset

For the dataset, we used English speech corpus LibriSpeech corpus [36] as a task to evaluate our proposed model model. Due to time and resource 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. 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 Kaldi's feature extractor [37] and normalized them to have zero mean and unit variance.

6.2 Model

First, we will briefly introduce the ASR system that we use in this experiment. Our baseline ASR model is based on "Deep Speech 2" architecture [38]. DeepSpeech2 model consists multiple different type of layers, including convolutional layer, recurrent bidirectional GRU layer, and fully connected layer. Given an input speech $X^{(i)}$ and transcription $Y^{(i)}$ sampled from the training set $\mathcal{D} = \{(X^{(1)}, Y^{(1)}), (X^{(2)}, Y^{(2)}), \ldots\}$, a single speech utterance is represented as a matrix $X^{(i)} \in \mathbb{R}^{S \times D}$ where T is the length of speech utterance and D is the feature dimension. A transcription output $Y^{(i)} = [y_1, y_2, \ldots, y_S]$ is a sequence of phoneme or grapheme plus a blank character with length T.

We illustrated the model architecture on the left side of Fig. 13. First, the speech features X are projected by one or more 1D convolution (convolution over the time dimension) and transformed by a nonlinear activation function ReLU:

$$h_s^l = f(\mathcal{W}^l \circledast h_{s-c:s+c}^{l-1}) \tag{62}$$

where $\mathcal{W}^l$ is a convolution filter weight at layer-l, c is the context window size and $f(x) = max(0, x)$ is a rectifier linear unit (ReLU) activation function. After that, the output from convolutional layers $h_{1:S}^l$ is fed into multiple bidirectional GRU layers:

$$\overrightarrow{h_t^l} = \text{GRU}(h_t^{l-1}, \overrightarrow{h_{t-1}^l}) \tag{63}$$

$$\overleftarrow{h_t^l} = \text{GRU}(h_t^{l-1}, \overleftarrow{h_{t+1}^l}). \tag{64}$$

The result from left-to-right GRU $\overrightarrow{h_t^l}$ is combined with the $\overleftarrow{h_t^l}$ by sum between two hidden states $h_t^l = \overrightarrow{h_t^l} + \overleftarrow{h_t^l}$. Next, the result from bidirectional GRU are fed into one or more fully connected layers, followed by nonlinear activation function ReLU:

$$h_t^l = f(W^l h_t^{l-1} + b^l). \tag{65}$$

Table 2 Librispeech dataset information

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

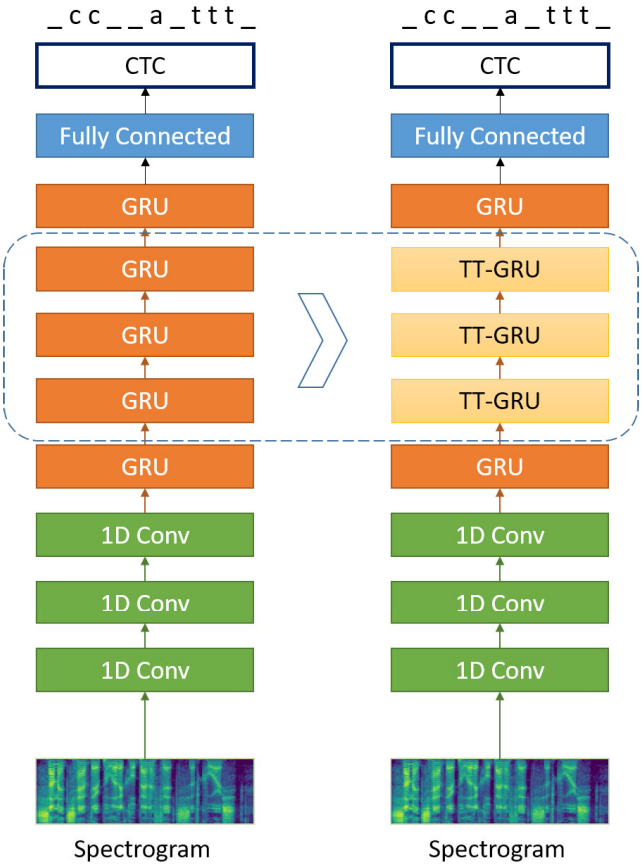

Fig. 13 We replace multiple layers of bidirectional GRU with TT-GRU

The last output from our model h_t^L are used to represent the label probability:

$$P(y_t = c|X) = \frac{\exp(w_c^L * h_t^{L-1} + b_c)}{\sum_k \exp(w_k^L * h_t^{L-1} + b_k)}. \tag{66}$$

To train this model, we use Connectionist Temporal Classification (CTC) loss function [39].

We perform the TT-format representation on the bidirectional GRU layers in the middle of DeepSpeech2 architecture. In Fig. 13, we mark the layer that we change from GRU to TT-GRU.

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 optimizer. However, our TT-GRU could not converge with SGD, and thus we used the Adam algorithm to optimize the TT-GRU model parameters. We evaluate ASR performance by calculating the normalized edit-distance between the generated transcription and the ground

Table 3 Parameters, Compression Rate, Validation CER and Test CER on Librispeech dataset

Model	Params	Compr.	Val CER	Test CER
GRU-H1510	13M	100	20.03%	20.62%
TTGRU				
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%

truth. Here, we use character error-rate (CER) because our output token are set of alphabet letters (a-z) and space character.

6.3 Result and Discussion

Table 3 shows the performance of the proposed TT-GRU in comparison with the baseline (uncompressed) GRU. 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. The best system of our proposed model could drastically reduce the parameters from 13 millions to 37,000. 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. Related Work

Compressing parameters on neural network architecture has become an interesting topic over the past several years due to the increased complexity of neural networks. The number of parameters and processing times has also grown tremendously along with their performance. A number of researchers comes up with many different ways to tackle this problem.

Ba et al. [10] and Hinton et al. [9] "distilled" the knowledge from a deep neural network into a shallow neural network. First, they trained a state-of-the-art model with a deep and complex neural network using the original dataset and hard label as the target. After that, they reused the trained deep neural network by extracting output from the softmax layer and used them as the output target for a shallow neural network. By training the shallow network with a soft target, they achieved a better performance than the model trained using hard target labels. Recently, Tang et al. [11] utilized a similar approach for training RNN with a trained DNN. However, they had to train two different neural networks and built different structures to transfer the knowledge from bigger models.

From the probabilistic perspective, Graves et al. [40] proposed a variational inference method for learning the mean and variance of Gaussian distribution for each weight parameter. They reformulated the variational inference as the optimization of a Minimum Description Length [41]. By modeling each weight parameter, they learned the importance of each weight in regard to the model. After the training process was finished, they pruned the parameters by removing the weight that has a high probability to be zero. However, they still needed large matrix multiplication and represented their model in dense weight matrix, and thus the algorithmic and memory complexity remained the same as in the original model. LeCun et al. [42] proposed a method to prune the weight with low-saliency based on their second derivatives.

Another approach to tackle the compression problem by a technical perspective is to limit the precision for weight parameters. Gupta et al. [43] and Courbariaux et al. [44] minimized the performance loss while using fewer bits (e.g., 16 bits) to represent floating points. Courbariaux et al. [45] proposed BinaryConnect to constrain the weight possible values to -1 or $+1$. Han et al. [46] utilized a combination between pruning, quantization and compression by Huffman coding. Most of these ideas can be easily applied with our proposed model since several deep-learning frameworks have built-in low-precision floating point options [47], [48].

Model compression using low-rank matrix has also been reported [12], [49]. Both of these works showed that many weight parameters are significantly redundant, and by representing them as low-rank matrices, they reduced the number of parameters with only a small drop in accuracy. Recently, Lu et al. [50] used low-rank matrix ideas to reduce the number of parameters in an RNN. Novikov et al. [13] utilized TT-format to represent weight matrices on feedforward neural networks. From their empirical evaluation on DNN-based architecture, the feedforward layer represented by the TT-format has a far better compression ratio and smaller accuracy loss compared to the low-rank matrix approach. Tjandra et al. [15] and Yang et al. [51] utilized the TT-format to represent the RNN weight matrices. Based on the empirical results, TT-format are able to reduce the number of parameters significantly and retain the model performance at the same time. Recent work from [52] used block decompositions to represent the RNN weight matrices.

To the best of our knowledge, there are only a few research about compression on RNN models. In this work, we presented an RNN model by using CP decomposition, Tucker decomposition and TT-decomposition to re-parameterize the weight matrices into a low-rank tensor format. We also compared the performance to standard uncompressed RNNs with a greater number of parameters. We expect our model could minimize the number of parameters and preserved the performance simultaneously.

8. Conclusion

In this paper, we presented an efficient and compact RNN model based on tensor decomposition method. In this work,

we presented some alternatives for compressing RNN parameters with tensor decomposition methods. Specifically, we utilized CP-decomposition and Tucker decomposition to represent the weight matrices. For the experiment, first, we run our experiment on polyphonic music dataset with uncompressed GRU model and three tensor-based RNN models (CP-GRU, Tucker-GRU and TT-GRU). We compare the performance of between all tensor-based RNNs under various number of parameters. Based on our experiment results, we conclude that TT-GRU has better performances compared to other methods under the same number of parameters. To extend our result, we run another experiment which is speech recognition task. We modified DeepSpeech2 architecture and we were able to represent dense weight matrices inside the RNN layer with multiple low-rank tensors based on TT-format. 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.

Acknowledgments

Part of this work was supported by JSPS KAKENHI Grant Numbers JP17H06101 and JP17K00237. We also thank Takuma Mori for his supports and insightful discussions.

References

[1] J.L. Elman, "Finding structure in time," Cognitive science, vol.14, no.2, pp.179–211, 1990.
[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol.9, no.8, pp.1735–1780, 1997.
[3] 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.
[4] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, et al., "Deep speech 2: End-to-end speech recognition in English and Mandarin," arXiv preprint arXiv:1512.02595, 2015.
[5] Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," arXiv preprint arXiv:1609.08144, 2016.
[6] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv preprint arXiv:1409.0473, 2014.
[7] I. Sutskever, O. Vinyals, and Q.V. Le, "Sequence to sequence learning with neural networks," Advances in neural information processing systems, pp.3104–3112, 2014.
[8] M. Schuster, "Speech recognition for mobile devices at Google," Pacific Rim International Conference on Artificial Intelligence, pp.8–10, Springer, 2010.
[9] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv preprint arXiv:1503.02531, 2015.
[10] J. Ba and R. Caruana, "Do deep nets really need to be deep?," Advances in neural information processing systems, pp.2654–2662, 2014.
[11] Z. Tang, D. Wang, and Z. Zhang, "Recurrent neural network training with dark knowledge transfer," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.5900–5904, IEEE, 2016.
[12] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al., "Predicting parameters in deep learning," Advances in Neural Information Processing Systems, pp.2148–2156, 2013.
[13] A. Novikov, D. Podoprikhin, A. Osokin, and D.P. Vetrov, "Tensorizing neural networks," Advances in Neural Information Processing Systems, pp.442–450, 2015.
[14] I.V. Oseledets, "Tensor-train decomposition," SIAM Journal on Scientific Computing, vol.33, no.5, pp.2295–2317, 2011.
[15] A. Tjandra, S. Sakti, and S. Nakamura, "Compressing recurrent neural network with tensor train," 2017 International Joint Conference on Neural Networks (IJCNN), pp.4451–4458, IEEE, 2017.
[16] A. Tjandra, S. Sakti, and S. Nakamura, "Tensor decomposition for compressing recurrent neural network," 2018 International Joint Conference on Neural Networks (IJCNN), pp.1–8, July 2018.
[17] T. Mori, A. Tjandra, S. Sakti, and S. Nakamura, "Compressing end-to-end ASR networks by tensor-train decomposition," Interspeech 2018, 19th Annual Conference of the International Speech Communication Association, Hyderabad, India, 2-6 Sept. 2018, pp.806–810, 2018.
[18] A. Graves, A.r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.6645–6649, IEEE, 2013.
[19] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Trans. Neural Netw., vol.5, no.2, pp.157–166, 1994.
[20] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.
[21] Q.V. Le, N. Jaitly, and G.E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," arXiv preprint arXiv:1504.00941, 2015.
[22] J. Martens and I. Sutskever, "Learning recurrent neural networks with Hessian-free optimization," Proc. 28th International Conference on Machine Learning (ICML-11), pp.1033–1040, 2011.
[23] A. Graves, N. Jaitly, and A.r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," 2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp.273–278, IEEE, 2013.
[24] 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.
[25] 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.
[26] R.A. Harshman, "Foundations of the parafac procedure: Models and conditions for an "explanatory" multimodal factor analysis," 1970.
[27] H.A. Kiers, "Towards a standardized notation and terminology in multiway analysis," Journal of chemometrics, vol.14, no.3, pp.105–122, 2000.
[28] T.G. Kolda and B.W. Bader, "Tensor decompositions and applications," SIAM review, vol.51, no.3, pp.455–500, 2009.
[29] L.R. Tucker, "Some mathematical notes on three-mode factor analysis," Psychometrika, vol.31, no.3, pp.279–311, 1966.
[30] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," International Conference on Machine Learning, pp.1310–1318, 2013.
[31] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," Proc. International Conference on Artificial Intelligence and Statistics (AISTATS'10), Society for Artificial Intelligence and Statistics, 2010.
[32] N. Boulanger-lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," Proc. 29th International Conference on Machine Learning (ICML-12), ed. J. Langford and J. Pineau, New York, NY, USA, pp.1159–1166, ACM,

2012.
[33] A.L. Maas, A.Y. Hannun, and A.Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," Proc. 30th International Conference on Machine Learning (ICML-13).
[34] M. Bay, A.F. Ehmann, and J.S. Downie, "Evaluation of multiple-f0 estimation and tracking systems," 2009 International Society for Music Information Retrieval Conference (ISMIR), pp.315–320, 2009.
[35] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
[36] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.5206–5210, IEEE, 2015.
[37] 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," IEEE 2011 Workshop on Automatic Speech Recognition and Understanding, IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No.: CFP11SRW-USB.
[38] 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," International Conference on Machine Learning, pp.173–182, 2016.
[39] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," Proc. 23rd international conference on Machine learning, pp.369–376, ACM, 2006.
[40] A. Graves, "Practical variational inference for neural networks," Advances in Neural Information Processing Systems, pp.2348–2356, 2011.
[41] G.E. Hinton and D. Van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," Proc. sixth annual conference on Computational learning theory, pp.5–13, ACM, 1993.
[42] Y. LeCun, J.S. Denker, and S.A. Solla, "Optimal brain damage," Advances in neural information processing systems, pp.598–605, 1990.
[43] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," Proc. 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, pp.1737–1746, 2015.
[44] M. Courbariaux, J.P. David, and Y. Bengio, "Training deep neural networks with low precision multiplications," arXiv preprint arXiv:1412.7024, 2014.
[45] M. Courbariaux, Y. Bengio, and J.P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," Advances in Neural Information Processing Systems, pp.3123–3131, 2015.
[46] S. Han, H. Mao, and W.J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," arXiv preprint arXiv:1510.00149, 2015.
[47] 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, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.
[48] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," arXiv e-prints, vol.abs/1605.02688, May 2016.
[49] 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," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp.6655–6659, IEEE, 2013.
[50] Z. Lu, V. Sindhwani, and T.N. Sainath, "Learning compact recurrent neural networks," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp.5960–5964, IEEE, 2016.
[51] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," International Conference on Machine Learning, pp.3891–3900, 2017.
[52] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, "Learning compact recurrent neural networks with block-term tensor decomposition," Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.9378–9387, 2018.

Andros Tjandra received his B.E degree in Computer Science (cum laude) from Faculty of Computer Science, Universitas Indonesia, Indonesia in 2014. Later, he continued his study and obtained M.S (cum laude) in 2015 from the same faculty and university. At the moment, he is currently taking doctoral course in Graduate School of Information Science, Nara Institute of Technology, Japan. He is a student member of ASJ. His research interests include machine learning (deep learning), speech recognition, speech synthesis and natural language processing.

Sakriani Sakti received her B.E. degree in Informatics (cum laude) from Bandung Institute of Technology, Indonesia, in 1999. In 2000, she received DAAD-Siemens Program Asia 21st Century Award to study in Communication Technology, University of Ulm, Germany, and received her MSc degree in 2002. During her thesis work, she worked with Speech Understanding Department, DaimlerChrysler Research Center, Ulm, Germany. Between 2003–2009, she worked as a researcher at ATR SLC Labs, Japan, and during 2006–2011, she worked as an expert researcher at NICT SLC Groups, Japan. While working with ATR-NICT, Japan, she continued her study (2005–2008) with Dialog Systems Group University of Ulm, Germany, and received her PhD degree in 2008. She was actively involved in collaboration activities such as Asian Pacific Telecommunity Project (2003–2007), A-STAR and U-STAR (2006–2011). In 2009–2011, she served as a visiting professor of Computer Science Department, University of Indonesia (UI), Indonesia. From 2011, she has been an assistant professor at the Augmented Human Communication Laboratory, NAIST, Japan. She served also as a visiting scientific researcher of INRIA Paris-Rocquencourt, France, in 2015–2016, under "JSPS Strategic Young Researcher Overseas Visits Program for Accelerating Brain Circulation". She is a member of JNS, SFN, ASJ, ISCA, IEICE and IEEE. Her research interests include statistical pattern recognition, speech recognition, spoken language translation, cognitive communication, and graphical modeling framework.

Satoshi Nakamura is Professor of Graduate School of Information Science, Nara Institute of Science and Technology, Japan, Honorarprofessor of Karlsruhe Institute of Technology, Germany, and ATR Fellow. He received his B.S. from Kyoto Institute of Technology in 1981 and Ph.D. from Kyoto University in 1992. He was Associate Professor of Graduate School of Information Science at Nara Institute of Science and Technology in 1994–2000. He was Director of ATR Spoken Language Communication Research Laboratories in 2000–2008 and Vice president of ATR in 2007–2008. He was Director General of Keihanna Research Laboratories and the Executive Director of Knowledge Creating Communication Research Center, National Institute of Information and Communications Technology, Japan in 2009–2010. He is currently Director of Augmented Human Communication laboratory and a full professor of Graduate School of Information Science at Nara Institute of Science and Technology. He is interested in modeling and systems of speech-to-speech translation and speech recognition. He is one of the leaders of speech-to-speech translation research and has been serving for various speech-to-speech translation research projects in the world including C-STAR, IWSLT and A-STAR. He received Yamashita Research Award, Kiyasu Award from the Information Processing Society of Japan, Telecom System Award, AAMT Nagao Award, Docomo Mobile Science Award in 2007, ASJ Award for Distinguished Achievements in Acoustics. He received the Commendation for Science and Technology by the Minister of Education, Science and Technology, and the Commendation for Science and Technology by the Minister of Internal Affairs and Communications. He also received LREC Antonio Zampoli Award 2012. He has been Elected Board Member of International Speech Communication Association, ISCA, since June 2011, IEEE Signal Processing Magazine Editorial Board Member since April 2012, IEEE SPS Speech and Language Technical Committee Member since 2013, and IEEE Fellow since 2016.