Compressing Recurrent Neural Networks with Tensor Ring for Action Recognition

Yu Pan,[1] **Jing Xu,**[1] **Maolin Wang,**[1] **Jinmian Ye,**[1] **Fei Wang,**[2] **Kun Bai,**[3] **Zenglin Xu**[1*]

[1]University of Electronic Science and Technology of China, Sichuan, China
Emails:{ypyupan, xujing.may, morin.w98, jinmian.y, zenglin}@gmail.com
[2]Weill Cornell Medical College, Cornell University, New York, NY, USA
Email:few201@cornell.edu
[3]Mobile Internet Group, Tencent Inc., Shenzhen, Guangdong, China
Email: kunbai@tencent.com

Abstract

Recurrent Neural Networks (RNNs) and their variants, such as Long-Short Term Memory (LSTM) networks, and Gated Recurrent Unit (GRU) networks, have achieved promising performance in sequential data modeling. The hidden layers in RNNs can be regarded as the memory units, which are helpful in storing information in sequential contexts. However, when dealing with high dimensional input data, such as video and text, the input-to-hidden linear transformation in RNNs brings high memory usage and huge computational cost. This makes the training of RNNs very difficult. To address this challenge, we propose a novel compact LSTM model, named as TR-LSTM, by utilizing the low-rank tensor ring decomposition (TRD) to reformulate the input-to-hidden transformation. Compared with other tensor decomposition methods, TR-LSTM is more stable. In addition, TR-LSTM can complete an end-to-end training and also provide a fundamental building block for RNNs in handling large input data. Experiments on real-world action recognition datasets have demonstrated the promising performance of the proposed TR-LSTM compared with the tensor-train LSTM and other state-of-the-art competitors.

Introduction

Recurrent Neural Networks (RNNs) have achieved great success in analyzing sequential data in various applications, such as computer vision (Byeon et al. 2015; Liang et al. 2016; Theis and Bethge 2015), natural language processing, etc.. Thanks to the ability in capturing long-range dependencies from input sequences (Rumelhart, Hinton, and Williams 1988; Sutskever, Vinyals, and Le 2014). To address the gradient vanishing issue which often leads to the failure of long-term memory in vanilla RNNs, advanced variants such as Gate Recurrent Unit (GRU) and Long-Short Term Memory (LSTM) have been proposed and applied in many learning tasks (Byeon et al. 2015; Liang et al. 2016; Theis and Bethge 2015).

Despite the success, LSTMs and GRUs suffer from the huge number of parameters, which makes the training process notoriously difficult and easily over-fitting. In particular, in the task of action recognition from videos, a video frame usually forms a high-dimensional input, which makes

the size of the input-to-hidden matrix extremely large. For example, in the UCF11 (Liu, Luo, and Shah 2009), a video action recognition dataset, an RGB video clip is a frame with a size of $160 \times 120 \times 3$, and the dimension of the input vector fed to the vanilla RNN can be over 57,000. Assume that the length of hidden layer vector is 256. Then, the input-to-hidden layer matrix has a load of parameters up to 14 millions. Although a pre-processing feature extraction step via deep convolutional neural networks can be utilized to obtain static feature maps as inputs to RNNs (Donahue et al. 2015), the over-parametric problem is still not fully solved.

A promising direction to reduce the parameter size is to explore the low-rank structures in the weight matrices. Inspired from the success of tensor decomposition methods in CNNs (Novikov et al. 2015; Li et al. 2017), various tensor decomposition methods have been explored in RNNs (Yang, Krompass, and Tresp 2017; Ye et al. 2018). In particular, in (Yang, Krompass, and Tresp 2017), the tensor train (TT) decomposition has been applied to RNNs in an end-to-end way to replace the input-to-hidden matrix, and achieved state-of-the-art performance in finding the low-rank structure in RNNs. However, the restricted setting on the ranks and the restrained order of core tensors makes TT-RNN models sensitive to parameter selection. In detail, the optimal setting of TT-ranks is that they are small in the border cores and large in middle cores, e.g., like an olive (Zhao et al. 2016).

To address this issue, we propose to use the tensor ring decomposition (TRD) (Zhao et al. 2016) to extract the low-rank structure of the input-to-hidden matrix in RNNs. Specifically, the input-to-hidden matrices are reshaped into a high-dimensional tensor and then factorized using TRD. Since this corresponding tensor ring layer automatically models the inter-parameter correlations, the number of parameters can be much smaller than the original size of the linear projection layer in standard RNNs. In this way, we present a new TR-RNN model with a similar representation power but with several orders of fewer parameters. In addition, since TRD can alleviate the strict constraints in tensor train decomposition via interconnecting the first and the last core tensors circularly (Zhao et al. 2016), we expect TR-RNNs to have more expressive power. It is important to note that the tensor ring layer can be optimized in an end-to-end training, and can also be utilized as a building block into current LSTM variants. For illustration, we implement an

LSTM with the tensor ring layer, named as TR-LSTM.[1]

We have conducted empirical evaluations on two real-world action recognition datasets, i.e., UCF11 and HMDB51 (Kuehne et al. 2011). For a fair comparison with standard LSTM and TT-LSTM (Yang, Krompass, and Tresp 2017) and BT-LSTM (Ye et al. 2018), we conduct experiments in an end-to-end training. As shown in Figure 7, the proposed TR-LSTM has obtained an accuracy value of 0.869, which outperforms the results of standard LSTM (i.e., 0.681), TT-LSTM (i.e.,0.803), and BT-LSTM (i.e., 0.856). Meanwhile, the compression ratio over the standard LSTM is over 34,000, which is also much higher than the compression ratio given by TT-LSTM and BT-LSTM. Moreover, with the output by pre-trained CNN as the input to LSTMs, TR-LSTM has outperformed most previous competitors including LSTM, TT-LSTM, BT-LSTM, and others recently proposed action recognition methods. Since the TR layer can be used as a building block to other LSTM-based approaches, such as the two-stream LSTM (Gammulle et al. 2017), we believe the proposed TR decomposition can be a promising approach for action recognition, by considering the tricks used in other state-of-the-art methods.

Model

To handle the high dimensional input of RNNs, we introduce the tensor ring decomposition to represent the input-to-hidden layer in RNNs in a compact structure. In the following, we first present preliminaries and background of tensor decomposition, including graphical illustrations of the tensor train decomposition and the tensor ring decomposition, followed by our proposed LSTM model, namely, TR-LSTM.

Preliminaries and Background

Notation In this paper, a d-order tensor, e.g., $\boldsymbol{\mathcal{D}} \in \mathbb{R}^{L_1 \times L_2 \cdots \times L_d}$ is denoted by a boldface Euler script letter. With all subscripts fixed, each element of a tensor is expressed as: $\boldsymbol{\mathcal{D}}_{l_1,l_2,\ldots l_d} \in \mathbb{R}$. Given a subset of subscripts, we can get a sub-tensor. For example, given a subset $\{L_1 = l_1, L_2 = l_2\}$, we can obtain a sub-tensor $\boldsymbol{\mathcal{D}}_{l_1,l_2} \in \mathbb{R}^{L_3 \cdots \times L_d}$. Specifically, we denote a vector by a bold lowercase letter, e.g., $\mathbf{v} \in \mathbb{R}^L$, and matrices by bold uppercase letters, e.g., $\mathbf{M} \in \mathbb{R}^{L_1 \times L_2}$. We regard vectors and matrices as 1-order tensors and 2-order tensors, respectively. Figure 1 draws the tensor diagrams presenting the graphical notations and the essential operations.

Tensor Contraction Tensor contraction can be performed between two tensors if some of their dimensions are matched. For example, given two 3-order tensors $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{L_1 \times L_2 \times L_3}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$, when $L_3 = J_1$, the contraction between these two tensors result in a tensor with the size of $L_1 \times L_2 \times J_2 \times J_3$, where the matching dimension is reduced, as shown in Equation (1):

$$(\boldsymbol{\mathcal{A}}\boldsymbol{\mathcal{B}})_{l_1,l_2,j_2,j_3} = \boldsymbol{\mathcal{A}}_{l_1,l_2}\boldsymbol{\mathcal{B}}_{j_2,j_3} = \sum_{p=1}^{L_3} \boldsymbol{\mathcal{A}}_{l_1,l_2,p}\boldsymbol{\mathcal{B}}_{p,j_2,j_3} \quad (1)$$

---

[1]Note that the tensor ring layer can also be plugged in the vanilla RNN and GRU.

(a) a $L_1 \times L_2$ matrix   (b) matrix contraction

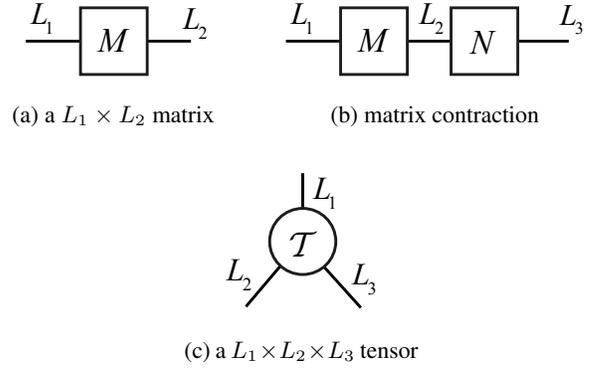(c) a $L_1 \times L_2 \times L_3$ tensor

Figure 1: Tensor diagrams. (a) shows the graphical representation of the matrix $\mathbf{M} \in \mathbb{R}^{L_1 \times L_2}$, where L_1 and L_2 denote the matrix size. A matrix is represented by a rectangular node. (b) demonstrates the contraction between two matrices(tensors), which is represented by an axis connecting them together and the contraction between $\mathbf{M}$ and $\mathbf{N}$ resulting in a new matrix with shape $\mathbb{R}^{L_1 \times L_3}$. (c) presents the graphical notation of a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{L_1 \times L_2 \times L_3}$.
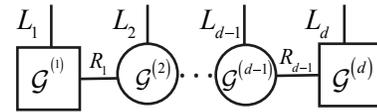

Figure 2: Tensor Train Decomposition: Noted that in tensor train the rank R_0 and R_d are constrained to 1, so the first and the last core are matrices while the inner cores are 3-order tensors.

Tensor Train Decomposition Through the tensor train decomposition (TTD), a high-order tensor can be decomposed as the product of a sequence of low-order tensors. For example, a d-order tensor $\boldsymbol{\mathcal{D}} \in \mathbb{R}^{L_1 \times L_2 \cdots \times L_d}$ can be decomposed as follows:

$$\boldsymbol{\mathcal{D}}_{l_1,l_2,\ldots,l_d} \overset{TTD}{=} \boldsymbol{\mathcal{G}}^{(1)}_{l_1}\boldsymbol{\mathcal{G}}^{(2)}_{l_2}\ldots\boldsymbol{\mathcal{G}}^{(d)}_{l_d}$$
$$= \sum_{r_1,r_2,\ldots,r_{d-1}} \boldsymbol{\mathcal{G}}^{(1)}_{r_0,l_1,r_1}\ldots\boldsymbol{\mathcal{G}}^{(d)}_{r_{d-1},l_d,r_d} \quad (2)$$

where each $\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{k-1} \times L_k \times R_k}$ is called a core tensor. The tensor train rank, shorted as TT-rank, $[R_0, R_1, R_2, \ldots, R_d]$, for each $k \in \{0, 1, \ldots, d\}, 0 < r_k \leq R_k$, corresponds to the complexity of tensor train decomposition. Generally, in tensor train decomposition, the constraint $R_0 = R_d = 1$ should be satisfied and other ranks are chosen manually. Figure 2 illustrates the form of tensor train decomposition.

Tensor Ring Decomposition The main drawback in tensor train decomposition is the limit of the ranks' setting, which hinders the representation ability and the flexibility of the TT-based models. At the same time, a strict order must be followed when multiplying TT cores, so that the alignment of the tensor dimensions is extremely important in obtaining

4684

(a) TRD in a ring form
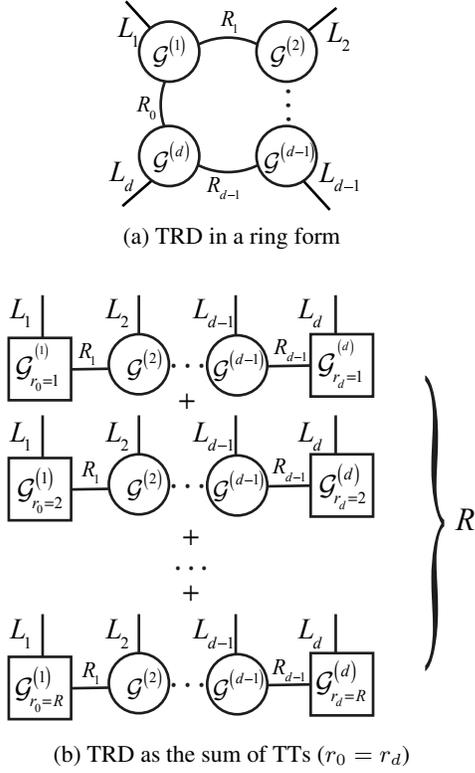

(b) TRD as the sum of TTs ($r_0 = r_d$)

Figure 3: Two representations of tensor ring decomposition (TRD). In Figure 3(a), TRD is expounded in the traditional way: the core tensors are multiplied one by one, and form a ring structure. In Figure 3(b), TRD is illustrated in an alternative way: the summation of a series of tensor trains. By fixing the subscript r_0 of $\mathbf{\mathcal{G}}^{(1)}$ and r_d of $\mathbf{\mathcal{G}}^{(d)}$: $r_0 = r_d = k$, where $k \in \{1, 2, \ldots, R\}$, both $\mathbf{\mathcal{G}}^{(1)}$ and $\mathbf{\mathcal{G}}^{(d)}$ are divided into R matrices.

the optimized TT cores, but it is still a challenging issue in finding the best alignment(Zhao et al. 2016).

In the tensor ring decomposition(TRD), an important modification is interconnecting the first and the last core tensors circularly and constructing a ring-like structure to alleviate the aforementioned limitations of the tensor train. Formally, we set $R_0 = R_d = R$ and $R \geq 1$, and conduct the decomposition as:

$$\mathbf{\mathcal{D}}_{l_1, l_2, \ldots, l_d} \overset{TRD}{=} \sum_{r_0 = r_d, r_2, \ldots, r_{d-1}} \mathbf{\mathcal{G}}^{(1)}_{r_0, l_1, r_1} \mathbf{\mathcal{G}}^{(2)}_{r_1, l_2, r_2} \cdots \mathbf{\mathcal{G}}^{(d)}_{r_{d-1}, l_d, r_d} \quad (3)$$

For a d-order tensor, by fixing the index k where $k \in \{1, 2, \ldots, R\}$, the first order of the beginning core tensor $\mathbf{\mathcal{G}}^{(1)}_{r_0 = k}$ and the last order of the ending core tensor $\mathbf{\mathcal{G}}^{(d)}_{r_d = k}$, can be reduced to matrices. Thus, along each of the R slices of $\mathbf{\mathcal{G}}^{(1)}$, we can separate the tensor ring structure as a summation of R of tensor trains. For example, by fixing $r_0 = r_d = k$, the product of $\mathbf{\mathcal{G}}^{(1)}_{k, l_1} \mathbf{\mathcal{G}}^{(2)}_{l_2} \ldots \mathbf{\mathcal{G}}^{(d)}_{l_d, k}$ has the form tensor train decomposition. Therefore, the tensor ring model

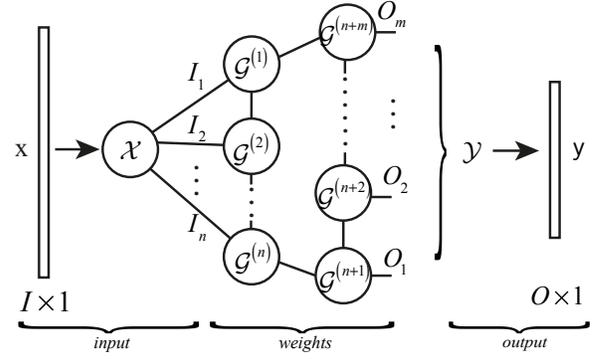Figure 4: TRL: $\mathbf{\mathcal{X}}$ represents the input tensor with shape $\mathbb{R}^{I_1 \times I_2 \times, \ldots, \times I_n}$ after reshaping the input vector $\mathbf{x} \in \mathbb{R}^{I \times 1}$. By performing the multiplication operation shown in Equation (6) with the weights in TRD form, the output tensor $\mathbf{\mathcal{Y}}$ with shape $\mathbb{R}^{O_1 \times O_2 \times, \ldots, \times O_m}$ can be obtained. Then, after transforming $\mathbf{\mathcal{Y}}$ into vector, we can get the final output vector $\mathbf{y} \in \mathbb{R}^O$.

is essentially the linear combination of R different tensor train models. Figure 3 demonstrates the tensor ring structure, and the alternative interpretation as a summation of multiple tensor train structures.

TR-RNN model

The core conception of our model is elaborated in this section. By transforming the input-to-hidden weight matrices in TR form, and applying them into RNN and its variants, we get our TR-RNN models.

Tensorizing x, y and W Without loss of generality, we tensorize the input vector $\mathbf{x} \in \mathbb{R}^I$, output vector $\mathbf{y} \in \mathbb{R}^O$, and weight matrix $\mathbf{W} \in \mathbb{R}^{I \times O}$ into tensors $\mathbf{\mathcal{X}}$, $\mathbf{\mathcal{Y}}$, and $\mathbf{\mathcal{W}}$, shown in Equation (4):

$$\mathbf{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times, \ldots, \times I_n}, \mathbf{\mathcal{Y}} \in \mathbb{R}^{O_1 \times O_2 \times, \ldots, \times O_m}$$

$$\mathbf{\mathcal{W}} \in \mathbb{R}^{I_1 \times I_2 \times, \ldots, \times I_n \times O_1 \times O_2, \ldots, \times O_m} \quad (4)$$

where

$$\prod_{i=1}^{n} I_i = I, \quad \prod_{j=1}^{m} O_j = O$$

Decomposing W For an n-order input and m-order output, we decompose the weight tensor into the form of TRD with $n + m$ core tensors multiplied one by one, each of which is corresponding to an input dimension or an output dimension, referring to Equation (5). Without loss of generality, the core tensors corresponding to the input dimensions and output dimensions are grouped respectively, as shown in Figure 4.

$$TRD(\mathbf{\mathcal{W}})_{i_1, \ldots, i_n, o_1, \ldots, o_m} = \sum_{r_0, \ldots, r_n, r_{n+1}, \ldots, r_{n+m-1}}$$

$$\mathbf{\mathcal{G}}^{(1)}_{r_0, i_1, r_1} \ldots \mathbf{\mathcal{G}}^{(n)}_{r_{n-1}, i_n, r_n} \mathbf{\mathcal{G}}^{(n+1)}_{r_n, o_1, r_{n+1}} \ldots \mathbf{\mathcal{G}}^{(n+m)}_{r_{n+m-1}, o_m, r_0} \quad (5)$$

The tensor contraction from input to hidden layer in TR form is shown in Equation (6). We multiply the input tensor with input core tensors and output core tensors sequentially. The complexity analysis of forward and backward process is elaborated in the appendix.

$$\boldsymbol{\mathcal{Y}}_{o_1,o_2,\ldots,o_m} = \sum_{i_1,\ldots,i_n} \boldsymbol{\mathcal{X}}_{i_1,\ldots,i_n} TRD(\boldsymbol{\mathcal{W}})_{i_1,\ldots,i_n,o_1,\ldots,o_m}$$

(6)

Compared with the redundant input-to-hidden weight matrix, the compression ratio in TR form is shown in Equation (7).

$$C_{TRD} = \frac{\prod_{i=1}^{i=n} I_i \prod_{j=1}^{j=m} O_j}{\sum_{i=1}^{n} R_{i-1} I_i R_i + \sum_{j=1}^{m} R_{n+j-1} O_j R_{n+j}}$$

(7)

Tensor Ring Layer (TRL)   After reshaping the input vector $\mathbf{x}$ and the weight matrix $\mathbf{W}$ into tensor, and decomposing weight tensor into TR representation, we can get the output tensor $\boldsymbol{\mathcal{Y}}$ by manipulating $\boldsymbol{\mathcal{W}}$ and $\boldsymbol{\mathcal{X}}$. The final output vector $\mathbf{y}$ can be obtained by reshaping the output tensor $\boldsymbol{\mathcal{Y}}$ into vector. Because the weight matrix is factorized with TRD, we denote the whole calculation from the input vector $\mathbf{x}$ to output vector $\mathbf{y}$ as tensor ring layer (TRL):

$$\mathbf{y} = TRL(\mathbf{W}, \mathbf{x}) \tag{8}$$

which is illustrated in Figure 4.

TR-RNN   By replacing the multiplication between weight matrix $\mathbf{W}_{hx}$ and input vector $\mathbf{x}$ with TRL in vanilla RNN. We get our TR-RNN model. The hidden state at time t can be expressed as:

$$\mathbf{h}_t = \sigma(TRL(\mathbf{W}_{hx}, \mathbf{x}_t) + \mathbf{U}_{hh}\mathbf{h}_{t-1} + \mathbf{b}) \tag{9}$$

where $\sigma(\cdot)$ denotes the sigmoid function and the hidden state is denoted by $\mathbf{h}_t$. The input-to-hidden layer weight matrix is denoted by $\mathbf{W}_{hx}$, and $\mathbf{U}_{hh}$ denotes the hidden-to-hidden layer matrix.

TR-LSTM   By applying TRL to the standard LSTM, which is the state-of-the-art variant of RNN, we can get the TR-LSTM model as follows.

$$\begin{aligned}
\mathbf{k}_t &= \sigma(TRL(\mathbf{W}_{kx}, \mathbf{x}_t) + \mathbf{U}_{kh}\mathbf{h}_{t-1} + \mathbf{b}_k) \\
\mathbf{f}_t &= \sigma(TRL(\mathbf{W}_{fx}, \mathbf{x}_t) + \mathbf{U}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(TRL(\mathbf{W}_{ox}, \mathbf{x}_t) + \mathbf{U}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{g}_t &= \tanh(TRL(\mathbf{W}_{gx}, \mathbf{x}_t) + \mathbf{U}_{gh}\mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{k}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

(10)

where $\odot$, $\sigma(\cdot)$ and $\tanh(\cdot)$ denote the element-wise product, the sigmoid function and the hyperbolic function, respectively. The weight matrices $\mathbf{W}_{*x}$ (where $*$ can be k, f, o, or g) denote the mapping from the input to hidden matrix, for the input gate $\mathbf{k}_t$, the forget gate $\mathbf{f}_t$, the output gate $\mathbf{o}_t$, and the cell update vector $\mathbf{c}_t$, respectively. The weight matrice $\mathbf{U}_{*h}$ are defined similarly for the hidden state $\mathbf{h}_{t-1}$.

Remark. As shown in Equation (6) and demonstrated in Figure 4, the multiplication between the input tensor

data $\mathcal{X}$ and the input core tensors $\mathcal{G}^{(i)}$ (for $i = 1, \ldots, n$) will produce a hidden matrix in the size of $R_0 \times R_n$. It is important to note that the size of the hidden matrix is much smaller than the original data size. In some sense, the "compressed" hidden matrix can be regarded as the information bottleneck (Tishby, Pereira, and Bialek 2000; Shwartz-Ziv and Tishby 2017), which seeks to achieve the balance between maximally compressing the input information and preserving the prediction information of the output. Thus the proposed TR-LSTM has high potentials to reduce the redundant information in the high-dimensional input while achieving good performance compared with the standard LSTM.

Experiments

To evaluate the proposed TR-LSTM model, we first design a synthetic experiment to validate the advantage of tensor ring decomposition over the tensor train decomposition. Through two real-world action recognition datasets, i.e., UCF11(YouTube action dataset) (Liu, Luo, and Shah 2009) and HMDB51 (Kuehne et al. 2011), we evaluate our model from two settings: (1) end-to-end training, where video frames are directly fed into the TR-LSTM; and (2) pre-training to obtain features prior to LSTMs, where a pre-trained CNN was used to extract meaningful low-dimensional features and then forwarded these features to the TR-LSTM. For a fair comparison, we first compare our proposed method with the standard LSTM and previous low-rank decomposition methods, and then with the state-of-the-art action recognition methods.

Synthetic Experiment

To verify the effectiveness of tensor decomposition methods in recover the original weights, we design a synthetic dataset. Given a low-rank weight matrix $\mathbf{W} \in \mathbb{R}^{81 \times 81}$, which is illustrated in Figure 5(a). We first sample 3200 examples, and each dimension follows a normal distribution, i.e., $\mathbf{x} \sim \mathcal{N}(0, 0.5\mathbf{I})$ where $\mathbf{I} \in \mathbb{R}^{81}$ is the identity matrix. We then calculate $\mathbf{y}$ according to $\mathbf{y} = \mathbf{W}\mathbf{x} + \epsilon$ for each $\mathbf{x}$ where $\epsilon \sim \mathcal{N}(0, \sigma^2\mathbf{I})$ is a random Gaussian noise and σ^2 is the variance. Since the $\mathbf{y}$ is generated from $\mathbf{x}$, the recovered weight matrix should be similar to the ground truth. We use the root mean square error (RMSE) to measure the performance. Should be noted that since the purpose of this experiment is to provide a qualitative and intuitive comparison, we do not add any regularization to the models.

Based on the input data and responses, we estimate the weight matrix $\mathbf{W}$ by running the linear regression, tensor train decomposition, and tensor ring decomposition, respectively. For tensor train and tensor ring, we first reshape input data to a tensor of $3 \times 3 \times 3 \times 3$, and reshape the weight matrix to a tensor of the same size. For illustration, Figure 5 shows one of the recovered $\mathbf{W}$ (reshaped as a matrix) for the three models when the noise variance is set to 0.05 . Clearly, the proposed tensor ring model performs the best among the three models. As for the tensor train model, it is even worse than the linear regression model. We further illustrate the recovered error of $\mathbf{W}$ with different levels of

noises in Figure 6. It demonstrates that the weight recovered by the tensor ring model has the best tolerance with respect to various injected noises.
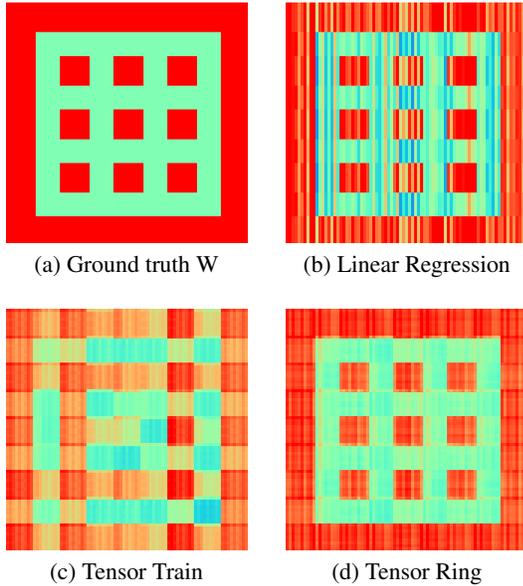

(a) Ground truth W (b) Linear Regression

(c) Tensor Train (d) Tensor Ring

Figure 5: The illustration on the ground truth $\mathbf{W}$ and the recovered weights from different models. The recovered RMSEs of the linear model, tensor train, and tensor ring, are 0.16, 0.18, and 0.09, respectively.
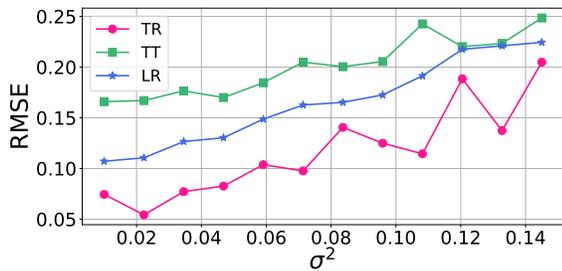

Figure 6: The illustration on how the RMSEs of the linear regression(LR), the tensor ring (TR) and the tensor train (TT) change with added noises.

Experiments on the UCF11 Dataset

The UCF11 dataset contains 1600 video clips of a resolution 320×240 divided into 11 action categories (e.g., basketball shooting, biking/cycling, diving, etc.). Each category consist of 25 groups of video, within more than 4 clips in one group. It is a challenging dataset due to large variations in camera motion, object appearance and pose, object scale, cluttered background, and so on.

In this part, we conduct two experiments described as "End-to-End Training" and "Pre-train with CNN" on this dataset. In the "End-to-End Training", we compare the proposed TR-LSTM model with other decomposition models
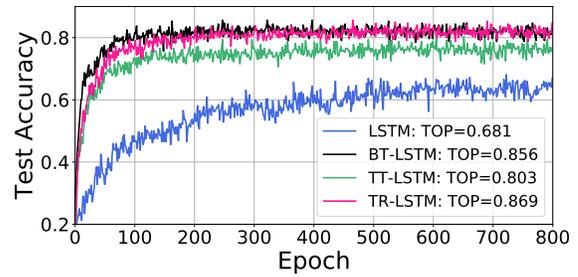
(eg. TT-LSTM (Yang, Krompass, and Tresp 2017) and BT-LSTM (Ye et al. 2018)) to show the superior performance. And in another experiment, we apply decomposition on a more general model, achieving a better performance with less parameters.
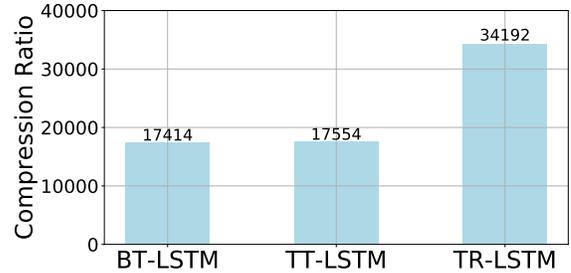
(a) Compression Ratio
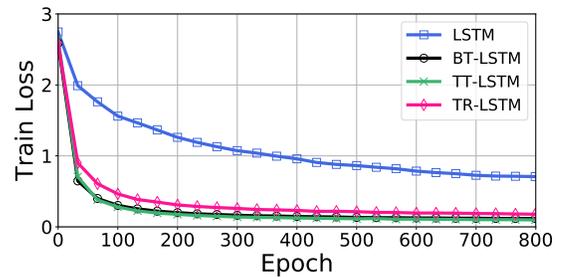
(b) Train Loss

(c) Test Accuracy

Figure 7: The results of "End-to-End Training" on UCF11 dataset. (a) shows the different compression ratio based on the vanilla LSTM. (b) and (c) shows the training and testing details.

End-to-End Training Recent years, some tensor decomposition models are proposed to classify videos like TT-LSTM (Yang, Krompass, and Tresp 2017), BT-LSTM (Ye et al. 2018) and others. For the reason that they use the end-to-end model for training, we set this experiment to compare with them. In this experiments, we scale down the original resolution to 160×120, and sample 6 frames from each video clip randomly as the input data. Since every frame is RGB, the input data vector at each step is $160 \times 120 \times 3 = 57600$, and there are 6 steps in every sam-

ple. We set the hidden layer as 256. So there should be a fully-connected layer of $4 \times 57600 \times 256 = 58982400$ parameters to achieve the mapping for the standard LSTM.

Table 1: Results of "End-to-End Training" on UCF11 reported in literature. TT-LSTM was reported in (Yang, Krompass, and Tresp 2017) while the BT-LSTM was reported in (Ye et al. 2018).

Method	#Params	Accuracy
LSTM	59M	0.697
TT-LSTM	3360	0.796
BT-LSTM	3387	0.853
TR-LSTM	1725	**0.869**

We compare our model with BT-LSTM and TT-LSTM, while using a standard LSTM as a baseline. The hyperparameters in BT-LSTM and TT-LSTM are set as announced in their papers. Figure 7(c) shows all decomposition methods converging faster than the LSTM. The accuracy of BT-LSTM is 0.856 which is much higher than TT-LSTM with 0.803 while the LSTM only gain an accuracy of 0.681. In our TR-LSTM, the shape of input tensor is $4 \times 2 \times 5 \times 8 \times 6 \times 5 \times 3 \times 2$, the output tensor's shape is set as $4 \times 4 \times 2 \times 4 \times 2$ and all the TR-ranks are set as 5 except $R_0 = R_d = 10$. Results are compared in Table 1. With 1725 parameters in our model, about half of TT-LSTM and BT-LSTM with parameters 3360 and 3387 respectively. We gain the top accuracy 0.869, showing the outstanding performance of our model in this experiment.

Table 2: The state-of-the-art performance on UCF11.

Method	Accuracy
(Hasan and Roy-Chowdhury 2014)	54.5%
(Liu, Luo, and Shah 2009)	71.2%
(Ikizler-Cinbis and Sclaroff 2010)	75.2%
(Liu, Shyu, and Zhao 2013)	76.1%
(Sharma, Kiros, and Salakhutdinov 2015)	85.0%
(Wang et al. 2011)	84.2%
(Sharma, Kiros, and Salakhutdinov 2015)	84.9%
(Cho et al. 2014)	88.0%
(Gammulle et al. 2017)	**94.6%**
CNN + LSTM	92.3%
CNN + TR-LSTM	93.8%

Pre-train with CNN Recently, some methods based on RNNs achieved higher accuracy by using the extracted feature as input vectors in computer vision (Donahue et al. 2015). Compared with using frames as input data, extracted features are more compact. But there is still some room for improving the ability of the models. The over-parametric problem is just partial solved. To get better performance, we use extracted features via the CNN model Inception-V3 as input data to LSTM.

We set the size of the hidden layer as $32 \times 64 = 2048$, which is consistent with the size of the output via Inception-V3. After using the extracted feature as the inputs of LSTM,

the accuracy of the vanilla LSTM attains 0.923. At the same time, the accuracy of our TR-LSTM model whose ranks are set as $40 \times 60 \times 48 \times 48$ achieves 93.8. By replacing the standard LSTM with our model, a compression ratio of 25 can be obtained. We compare some state-of-the-art methods in Table 2 on UCF11. The Two Stream LSTM(Gammulle et al. 2017) with highest accuracy has more than 141M parameters. The TR-LSTM can be used to replace the vanilla LSTMs in the Two Stream LSTM model to reduce the parameters.

Experiments on the HMDB51 Dataset

The HMDB51 dataset is a large collection of realistic videos from various sources, such as movies and web videos. The dataset is composed of 6766 video clips from 51 action categories.

Table 3: Comparison with state-of-the-art results on HMDB51. The best accuracy is 0.664 from the I3D model reported in (Carreira and Zisserman 2017), which bases on 3D ConvNets and is not RNN-based method.

Method	Accuracy
(Cai et al. 2014)	55.9%
(Wang and Schmid 2013)	57.2%
(Tran et al. 2015)	56.8%
(Feichtenhofer, Pinz, and Zisserman 2016)	56.8%
(Wang, Qiao, and Tang 2015)	63.2%
(Carreira and Zisserman 2017)	**66.4%**
(Ni et al. 2015)	65.5%
(Jain, Jegou, and Bouthemy 2013)	52.1%
(Zhu et al. 2013)	54.0%
CNN + LSTM	62.9%
CNN + TR-LSTM	63.8%

In this experiment, we still use extracted features as the input vector via Inception-V3 and reshape it into 64×32. We sample 12 frames from each video clip randomly and be processed through the CNN as the input data. The shape of hidden layer tensor is set as $32 \times 64 = 2048$. The ranks of our TR-LSTM are $40 \times 60 \times 48 \times 48$. Some of the state-of-the-art models like I3D (Carreira and Zisserman 2017) are presented in Table 3. The I3D model with highest accuracy based on 3D ConvNets, which is not RNN-based method, has 25M parameters, while the TR-LSTM model only has 0.7M parameters. The TR-LSTM gains a higher accuracy of 63.8% than the standard LSTM with a compressing ratio of 25.

Related Work

In the past decades, a number of variants of recurrent neural networks (RNNs) were proposed to capture sequential information more accurately (Liu et al. 2018b). However, when dealing with the large input data, in the field of computer vision, the input-to-hidden weight matrix owns loads of parameters. The limitation of computation resources and the severe over-fitting problem are emerging. Some methods used CNNs as feature extractors to pre-processing the input data into a more compact way (Donahue et al. 2015). These

methods have improved the classification accuracy, but the over-parametric problem is just still partially solved. In this work, we focus on designing low-rank structure to replace the redundant input-to-hidden weight matrix in RNNs, while compressing the whole model and maintaining the model performance.

The most straight-forward way to apply low-rank constraint is implementing matrix decomposition on weight matrices. Singular Value Decomposition (SVD) has been applied in convolutional neural networks to reduce parameters (Denton et al. 2014) but incurred a loss in model performance. Besides, the compression ratio is limited because of the rank in matrix decomposition still relatively large.

Compared with matrix decomposition, tensor decomposition (Xu, Yan, and Qi 2015; Zhe et al. 2016; Hao et al. 2018; He et al. 2018; Liu et al. 2018a) conducts data decomposition in a higher dimension, capturing higher-order correlations while maintaining several orders of fewer parameters (Kim et al. 2015; Novikov et al. 2015). Among these methods, (Lebedev et al. 2014) utilized CP decomposition to speed up convolution computation, which has the similar design philosophy with the widely-used depth-wise separable convolutions (Howard et al. 2017). However, the instability issue (de Silva and Lim 2008) hinders the low-rank CP decomposition from solving many important computer vision tasks. (Kim et al. 2015) used Tucker decomposition to decompose both the convolution layer and the fully connected layer, reducing run-time and energy significantly in mobile applications with minor accuracy drop. Block-Term tensor decomposition combines the CP and Tucker by summing up multiple Tucker models to overcome their drawbacks and has obtained a better performance in RNNs (Ye et al. 2018). However, the computation of the core tensor in the Tucker model is highly inefficient due to the complex tensor flatten and permutation operations. Recent years, the tensor train decomposition also used to substitute the redundant fully connected layer in both CNNs and RNNs (Novikov et al. 2015; Yang, Krompass, and Tresp 2017), preserving the performance while reducing the number of parameters significantly up to 40 times.

But tensor train decomposition has some limitations: 1) certain constraints for TT-rank, i.e., the ranks of the first and last factors are restricted to be 1, limiting its representation power and flexibility. 2) A strict order must be followed when multiplying TT cores, so that the alignment of the tensor dimensions is extremely important in obtaining the optimized TT cores, but it is still a challenging issue in finding the best alignment. In this paper, we use the Tensor Ring(TR) decomposition (Zhao et al. 2016) to overcome the drawbacks in TTD, while achieving more computation efficiency than BT decomposition.

Conclusion

In this paper, we applied TRD to plain RNNs to replace the over-parametric input-to-hidden weight matrix when dealing with high-dimensional input data. The low-rank structure of TRD can capture the correlation between feature dimensions with fewer orders of magnitude parameters. Our TR-LSTM model achieved best compression ratio with the highest classification accuracy on UCF11 dataset among other end-to-end training RNNs based on low-rank methods. At the same time, when processing the extracted feature through InceptionV3 as the input vector, our TR-LSTM model can still compress the LSTM while improving the accuracy. We believe that our models provide fundamental modules for RNNs, and can be widely used to handle large input data. In future work, since our models are easy to be extended, we want to apply our models to more advanced RNN structures (Gammulle et al. 2017) to get better performance.

Acknowledgments

References

Byeon, W.; Breuel, T. M.; Raue, F.; and Liwicki, M. 2015. Scene labeling with lstm recurrent neural networks. In *CVPR*, 3547–3555.

Cai, Z.; Wang, L.; Peng, X.; and Qiao, Y. 2014. Multi-view super vector for action recognition. In *CVPR*, 596–603. IEEE Computer Society.

Carreira, J., and Zisserman, A. 2017. Quo vadis, action recognition? A new model and the kinetics dataset. In *CVPR*, 4724–4733. IEEE Computer Society.

Cho, J.; Lee, M.; Chang, H. J.; and Oh, S. 2014. Robust action recognition using local motion and group sparsity. *Pattern Recognition 2014* 47(5):1813–1825.

de Silva, V., and Lim, L. 2008. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM J. Matrix Analysis Applications* 30(3):1084–1127.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 1269–1277.

Donahue, J.; Hendricks, L. A.; Guadarrama, S.; Rohrbach, M.; Venugopalan, S.; Darrell, T.; and Saenko, K. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2625–2634. IEEE Computer Society.

Feichtenhofer, C.; Pinz, A.; and Zisserman, A. 2016. Convolutional two-stream network fusion for video action recognition. In *CVPR*, 1933–1941. IEEE Computer Society.

Gammulle, H.; Denman, S.; Sridharan, S.; and Fookes, C. 2017. Two stream LSTM: A deep fusion framework for human action recognition. In *WACV*, 177–186. IEEE.

Hao, L.; Liang, S.; Ye, J.; and Xu, Z. 2018. Tensord: A tensor decomposition library in tensorflow. *Neurocomputing* 318:196–200.

Hasan, M., and Roy-Chowdhury, A. K. 2014. Incremental activity modeling and recognition in streaming videos. In *CVPR*, 796–803. IEEE.

He, L.; Liu, B.; Li, G.; Sheng, Y.; Wang, Y.; and Xu, Z. 2018. Knowledge base completion by variational bayesian neural tensor decomposition. *Cognitive Computation*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR* abs/1704.04861.

Ikizler-Cinbis, N., and Sclaroff, S. 2010. Object, scene and actions: Combining multiple features for human action recognition. In *ECCV*, 494–507. Springer.

Jain, M.; Jegou, H.; and Bouthemy, P. 2013. Better exploiting motion for better action recognition. In *CVPR 2013*, 2555–2562. IEEE Computer Society.

Kim, Y.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *CoRR* abs/1511.06530.

Kuehne, H.; Jhuang, H.; Garrote, E.; Poggio, T. A.; and Serre, T. 2011. HMDB: A large video database for human motion recognition. In *ICCV*, 2556–2563. IEEE Computer Society.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I. V.; and Lempitsky, V. S. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *CoRR* abs/1412.6553.

Li, G.; Ye, J.; Yang, H.; Chen, D.; Yan, S.; and Xu, Z. 2017. Bt-nets: Simplifying deep neural networks via block term decomposition. *CoRR* abs/1712.05689.

Liang, X.; Shen, X.; Xiang, D.; Feng, J.; Lin, L.; and Yan, S. 2016. Semantic object parsing with local-global long short-term memory. In *CVPR*, 3185–3193.

Liu, B.; He, L.; Li, Y.; Zhe, S.; and Xu, Z. 2018a. Neuralcp: Bayesian multiway data analysis with neural tensor decomposition. *Cognitive Computation*.

Liu, H.; He, L.; Bai, H.; Dai, B.; Bai, K.; and Xu, Z. 2018b. Structured inference for recurrent hidden semi-markov model. In *IJCAI*, 2447–2453.

Liu, J.; Luo, J.; and Shah, M. 2009. Recognizing realistic actions from videos "in the wild". In *CVPR*, 1996–2003. IEEE.

Liu, D.; Shyu, M.-L.; and Zhao, G. 2013. Spatial-temporal motion information integration for action detection and recognition in non-static background. In *IRI*, 626–633. IEEE.

Ni, B.; Moulin, P.; Yang, X.; and Yan, S. 2015. Motion part regularization: Improving action recognition via trajectory group selection. In *CVPR*, 3698–3706. IEEE Computer Society.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. In *NIPS*, 442–450.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5(3):1.

Sharma, S.; Kiros, R.; and Salakhutdinov, R. 2015. Action recognition using visual attention. *CoRR* abs/1511.04119.

Shwartz-Ziv, R., and Tishby, N. 2017. Opening the black box of deep neural networks via information. *CoRR* abs/1703.00810.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*, 3104–3112.

Theis, L., and Bethge, M. 2015. Generative image modeling using spatial lstms. In *NIPS*, 1927–1935.

Tishby, N.; Pereira, F. C. N.; and Bialek, W. 2000. The information bottleneck method. *CoRR* physics/0004057.

Tran, D.; Bourdev, L. D.; Fergus, R.; Torresani, L.; and Paluri, M. 2015. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*, 4489–4497. IEEE Computer Society.

Wang, H., and Schmid, C. 2013. Action recognition with improved trajectories. In *ICCV*, 3551–3558. IEEE Computer Society.

Wang, H.; Kläser, A.; Schmid, C.; and Liu, C. 2011. Action recognition by dense trajectories. In *CVPR*, 3169–3176. IEEE.

Wang, L.; Qiao, Y.; and Tang, X. 2015. Action recognition with trajectory-pooled deep-convolutional descriptors. In *CVPR*, 4305–4314. IEEE Computer Society.

Xu, Z.; Yan, F.; and Qi, Y. A. 2015. Bayesian nonparametric models for multiway data analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 37(2):475–487.

Yang, Y.; Krompass, D.; and Tresp, V. 2017. Tensor-train recurrent neural networks for video classification. In *ICML*, 3891–3900.

Ye, J.; Wang, L.; Li, G.; Chen, D.; Zhe, S.; Chu, X.; and Xu, Z. 2018. Learning compact recurrent neural networks with block-term tensor decomposition. In *CVPR*.

Zhao, Q.; Zhou, G.; Xie, S.; Zhang, L.; and Cichocki, A. 2016. Tensor ring decomposition. *CoRR* abs/1606.05535.

Zhe, S.; Zhang, K.; Wang, P.; Lee, K.; Xu, Z.; Qi, Y.; and Ghahramani, Z. 2016. Distributed flexible nonlinear tensor factorization. In *NIPS*, 920–928.

Zhu, J.; Wang, B.; Yang, X.; Zhang, W.; and Tu, Z. 2013. Action recognition with actons. In *ICCV*, 3559–3566. IEEE Computer Society.