Compressing Recurrent Neural Networks with Tensor Ring for Action Recognition

Yu Pan[1], **Jing Xu**[1], **Maolin Wang**[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, 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 unscalable and 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.

1 Introduction

Recurrent Neural Networks (RNNs) have achieved great success in analyzing sequential data in various applications, such as computer vision [1, 13, 19], natural language processing, etc.. 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.

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 [17, 12], various tensor decomposition methods have been explored in RNNs [24, 25]. And we propose to use the tensor ring decomposition (TRD) [26] to extract the low-rank structure of the input-to-hidden matrix in RNNs(the details of the TRD are in Appendix 5.1). For illustration, we implement the tensor ring layer on LSTM, replacing the over-parametric input to hidden matrices, named TR-LSTM. And, the tensor ring layer can also be plugged in the vanilla RNN and GRU.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

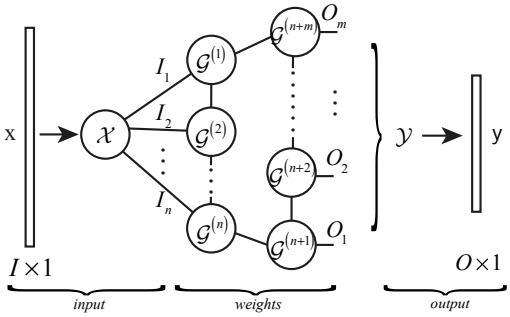


Figure 1: TRL: $\mathcal{X}$ represents the input tensor with shape $\mathbb{R}^{I_1 \times I_2 \times,\ldots,\times I_n}$ after reshaping the input vector $\mathbf{x} \in \mathbb{R}^{I \times 1}$. By performing the multiplication operation shown in Equation (8) with the weights in TRD form, the output tensor $\mathcal{Y}$ with shape $\mathbb{R}^{O_1 \times O_2 \times,\ldots,\times O_m}$ can be obtained. Then, after transforming $\mathcal{Y}$ into vector, we can get the final output vector $\mathbf{y} \in \mathbb{R}^{O \times 1}$.

2 Model

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

2.1 Tensor Ring Layer (TRL)

After reshaping the input vector $\mathbf{x}$ and the weight matrix $\mathbf{W}$ into tensor(introduced in Appendix 5.2.1), and decomposing weight tensor into TR representation(described in Appendix 5.2.2), we can get the output tensor $\mathcal{Y}$ by manipulating $\mathcal{W}$ and $\mathcal{X}$. After reshaping the output tensor, the final output vector $\mathbf{y}$ can be obtained. Because the weight matrix is factorized with TRD, we denote the whole calculation from input to output vector as tensor ring layer (TRL):

$$\mathbf{y} = TRL(\mathbf{W}, \mathbf{x}) \tag{1}$$

which is illustrated in Figure 1.The complexity analysis are shown in Appendix 5.2.3.

2.2 TR-LSTM

We can get our TR-LSTM model by applying TRL to LSTM, which is the state-of-the-art variant of RNN.

$$\begin{aligned}
\mathbf{k}_t &= \sigma(TRL(\mathbf{W}_{kx}, \mathbf{x}_t) + \mathbf{U}_{kh}\mathbf{h}_{t-1} + \mathbf{b}_k) \\
\mathbf{f}_t &= \sigma(TRL(\mathbf{W}_{fx}, \mathbf{x}_t) + \mathbf{U}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{o}_t &= \sigma(TRL(\mathbf{W}_{ox}, \mathbf{x}_t) + \mathbf{U}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{g}_t &= \tanh(TRL(\mathbf{W}_{gx}, \mathbf{x}_t) + \mathbf{U}_{gh}\mathbf{h}_{t-1} + \mathbf{b}_g) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{k}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{2}$$

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

3 Experiments

3.1 Experiments on the UCF11 Dataset

We conduct two experiments described as "End-to-End Training" and "Pre-train with CNN" on the UCF11 Dataset, where "End-to-End Training" means that video frames are directly fed into the TR-LSTM and "Pre-train with CNN" means the inputs of the TR-LSTM are extracted features via a pre-trained CNN. Details of this experiments can be found in Appendix 5.3.

Table 1: Results of "End-to-End Training" on UCF11 reported in literature.

Method	#Params	Accuracy
LSTM	59M	0.697
TT-LSTM [24]	3360	0.796
BT-LSTM [25]	3387	0.853
TR-LSTM	**1725**	**0.869**

End-to-End Training We compare our model with BT-LSTM [25] and TT-LSTM [24], while using a standard LSTM as a baseline. 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. We gain the top accuracy 0.869 with the least parameters of 1725, showing the outstanding performance of our model in this experiment.

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

Method	Accuracy
[9]	54.5%
[15]	71.2%
[10]	75.2%
[14]	76.1%
[18]	85.0%
[22]	84.2%
[18]	84.9%
[4]	88.0%
[8]	**94.6%**
CNN + LSTM	92.3%
CNN + TR-LSTM	93.8%

Table 3: Comparison with state-of-the-results on HMDB51. The best accuracy is 0.664 from the I3D model reported in [3], which used both image and optical flow information.

Method	Accuracy
[2]	55.9%
[21]	57.2%
[20]	56.8%
[7]	56.8%
[23]	63.2%
[3]	**66.4%**
[16]	65.5%
[11]	52.1%
[27]	54.0%
CNN + LSTM	62.9%
CNN + TR-LSTM	63.8%

Pre-train with CNN 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 with a compression ratio of 25. We compare some state-of-the-art methods in Table 2 on UCF11.

3.2 Experiments on the HMDB51 Dataset

In this experiment, we still use extracted features via Inception-V3 as the input vector 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 [3] are presented in Table 3. With a compressing ratio of 25, the TR-LSTM model still gains a higher accuracy of 63.8% than the standard LSTM.

4 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. 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 [8] to get better performance.

References

[1] Wonmin Byeon, Thomas M Breuel, Federico Raue, and Marcus Liwicki. Scene labeling with lstm recurrent neural networks. In *CVPR 2015*, pages 3547–3555, 2015.

[2] Zhuowei Cai, Limin Wang, Xiaojiang Peng, and Yu Qiao. Multi-view super vector for action recognition. In *CVPR 2014*, pages 596–603. IEEE Computer Society, 2014. doi: 10.1109/CVPR.2014.83. URL https://doi.org/10.1109/CVPR.2014.83.

[3] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *CVPR 2017*, pages 4724–4733. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.502. URL https://doi.org/10.1109/CVPR.2017.502.

[4] Jungchan Cho, Minsik Lee, Hyung Jin Chang, and Songhwai Oh. Robust action recognition using local motion and group sparsity. *Pattern Recognition 2014*, 47(5):1813–1825, 2014. doi: 10.1016/j.patcog.2013.12.004. URL https://doi.org/10.1016/j.patcog.2013.12.004.

[5] Andrzej Cichocki, Namgil Lee, Ivan V. Oseledets, Anh Huy Phan, Qibin Zhao, and Danilo P. Mandic. Low-rank tensor networks for dimensionality reduction and large-scale optimization problems: Perspectives and challenges PART 1. *CoRR*, abs/1609.00893, 2016. URL http://arxiv.org/abs/1609.00893.

[6] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Trevor Darrell, and Kate Saenko. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR 2015*, pages 2625–2634. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298878. URL https://doi.org/10.1109/CVPR.2015.7298878.

[7] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *CVPR 2016*, pages 1933–1941. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.213. URL https://doi.org/10.1109/CVPR.2016.213.

[8] Harshala Gammulle, Simon Denman, Sridha Sridharan, and Clinton Fookes. Two stream LSTM: A deep fusion framework for human action recognition. In *WACV 2017*, pages 177–186. IEEE, 2017. doi: 10.1109/WACV.2017.27. URL https://doi.org/10.1109/WACV.2017.27.

[9] Mahmudul Hasan and Amit K. Roy-Chowdhury. Incremental activity modeling and recognition in streaming videos. In *CVPR 2014*, pages 796–803. IEEE, 2014. doi: 10.1109/CVPR.2014.107. URL https://doi.org/10.1109/CVPR.2014.107.

[10] Nazli Ikizler-Cinbis and Stan Sclaroff. Object, scene and actions: Combining multiple features for human action recognition. In *ECCV 2010*, pages 494–507. Springer, 2010.

[11] Mihir Jain, Herve Jegou, and Patrick Bouthemy. Better exploiting motion for better action recognition. In *CVPR 2013*, pages 2555–2562. IEEE Computer Society, 2013. doi: 10.1109/CVPR.2013.330. URL https://doi.org/10.1109/CVPR.2013.330.

[12] Guangxi Li, Jinmian Ye, Haiqin Yang, Di Chen, Shuicheng Yan, and Zenglin Xu. Bt-nets: Simplifying deep neural networks via block term decomposition. *CoRR*, abs/1712.05689, 2017.

[13] Xiaodan Liang, Xiaohui Shen, Donglai Xiang, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with local-global long short-term memory. In *CVPR 2016*, pages 3185–3193, 2016.

[14] Dianting Liu, Mei-Ling Shyu, and Guiru Zhao. Spatial-temporal motion information integration for action detection and recognition in non-static background. In *IRI 2013*, pages 626–633. IEEE, 2013.

[15] Jingen Liu, Jiebo Luo, and Mubarak Shah. Recognizing realistic actions from videos "in the wild". In *CVPR 2009*, pages 1996–2003. IEEE, 2009.

[16] Bingbing Ni, Pierre Moulin, Xiaokang Yang, and Shuicheng Yan. Motion part regularization: Improving action recognition via trajectory group selection. In *CVPR 2015*, pages 3698–3706. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7298993. URL https://doi.org/10.1109/CVPR.2015.7298993.

[17] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *NIPS*, pages 442–450, 2015. URL http://papers.nips.cc/paper/5787-tensorizing-neural-networks.

[18] Shikhar Sharma, Ryan Kiros, and Ruslan Salakhutdinov. Action recognition using visual attention. *CoRR*, abs/1511.04119, 2015. URL http://arxiv.org/abs/1511.04119.

[19] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *NIPS 2015*, pages 1927–1935, 2015.

[20] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *ICCV 2015*, pages 4489–4497. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.510. URL https://doi.org/10.1109/ICCV.2015.510.

[21] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *ICCV 2013*, pages 3551–3558. IEEE Computer Society, 2013. doi: 10.1109/ICCV.2013.441. URL https://doi.org/10.1109/ICCV.2013.441.

[22] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Action recognition by dense trajectories. In *CVPR 2011*, pages 3169–3176. IEEE, 2011. doi: 10.1109/CVPR.2011.5995407. URL https://doi.org/10.1109/CVPR.2011.5995407.

[23] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *CVPR 2015*, pages 4305–4314. IEEE Computer Society, 2015. doi: 10.1109/CVPR.2015.7299059. URL https://doi.org/10.1109/CVPR.2015.7299059.

[24] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *ICML 2017*, pages 3891–3900, 2017.

[25] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In *CVPR 2018*, June 2018.

[26] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *CoRR*, abs/1606.05535, 2016. URL http://arxiv.org/abs/1606.05535.

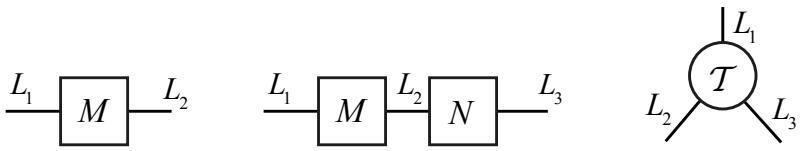
[27] Jun Zhu, Baoyuan Wang, Xiaokang Yang, Wenjun Zhang, and Zhuowen Tu. Action recognition with actons. In *ICCV 2013*, pages 3559–3566. IEEE Computer Society, 2013. doi: 10.1109/ICCV.2013.442. URL https://doi.org/10.1109/ICCV.2013.442.

5 Appendix

5.1 Preliminaries and Background

5.1.1 Notation

In this paper, a d-order tensor, e.g., $\mathcal{D} \in \mathbb{R}^{L_1 \times L_2 \cdots \times L_d}$ is denoted by a boldface Euler script letter. With all subscripts fixed, each element of a tensor is expressed as: $\mathcal{D}_{l_1,l_2,\ldots l_d} \in \mathbb{R}$. 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 $\mathcal{D}_{l_1,l_2} \in \mathbb{R}^{L_3 \cdots \times L_d}$. 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 2 draws the tensor diagrams presenting the graphical notations and the essential operations.



(a) a $L_1 \times L_2$ matrix (b) matrix contraction (c) a $L_1 \times L_2 \times L_3$ tensor

Figure 2: 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 $\mathcal{T} \in \mathbb{R}^{L_1 \times L_2 \times L_3}$.

5.1.2 Tensor Contraction

Tensor contraction can be performed between two tensors if some of their dimensions are matched. For example, given two 3-order tensors $\mathcal{A} \in \mathbb{R}^{L_1 \times L_2 \times L_3}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$, 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 (3):

$$(\mathcal{A}\mathcal{B})_{l_1,l_2,j_2,j_3} = \mathcal{A}_{l_1,l_2}\mathcal{B}_{j_2,j_3} = \sum_{p=1}^{L_3} \mathcal{A}_{l_1,l_2,p}\mathcal{B}_{p,j_2,j_3} \tag{3}$$

5.1.3 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 $\mathcal{D} \in \mathbb{R}^{L_1 \times L_2 \cdots \times L_d}$ can be decomposed as follows:

$$\begin{aligned} \mathcal{D}_{l_1,l_2,\ldots,l_d} &\overset{TTD}{=} \mathcal{G}^{(1)}_{l_1}\mathcal{G}^{(2)}_{l_2}\cdots\mathcal{G}^{(d)}_{l_d} \\ &= \sum_{r_1,r_2,\ldots,r_{d-1}} \mathcal{G}^{(1)}_{r_0,l_1,r_1}\cdots\mathcal{G}^{(d)}_{r_{d-1},l_d,r_d} \end{aligned} \tag{4}$$

where each $\mathcal{G}^{(k)} \in \mathbb{R}^{R_{k-1} \times L_k \times R_k}$ 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 3 illustrates the form of tensor train decomposition.

5.1.4 Tensor Ring Decomposition

The main drawback in tensor train decomposition is its constrained in ranks' setting, hindering the representation ability and the flexibility of the TT-based models. At the same time, a strict order must

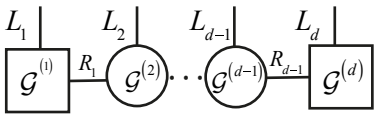
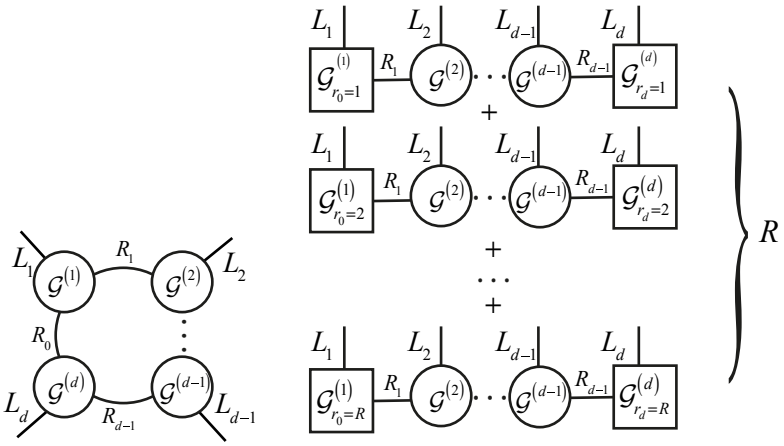


Figure 3: Tensor Train Decomposition: Noted that in tensor train the rank R_0 and R_d are constrained to 1, so the first and the last core are matrices while the inner cores are 3-order tensors.



(a) TRD in a ring form (b) TRD as the sum of TTs ($r_0 = r_d$)

Figure 4: Two representations of tensor ring decomposition (TRD). In Figure 4(a), TRD is expounded in the traditional way: the core tensors are multiplied one by one, and form a ring structure. In Figure 4(b), TRD is illustrated in an alternative way: the summation of a series of tensor trains. By fixing the subscript r_0 of $\mathcal{G}^{(1)}$ and r_d of $\mathcal{G}^{(d)}$: $r_0 = r_d = k$, where $k \in \{1, 2, \ldots, R\}$, each core tensor is divided into R matrices.

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[26].

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:

$$\mathcal{D}_{l_1,l_2,\ldots,l_d} \overset{TRD}{=} \sum_{r_0=r_d=1}^{R} \mathcal{G}^{(1)}_{r_0,l_1} \mathcal{G}^{(2)}_{l_2} \cdots \mathcal{G}^{(d)}_{l_d,r_d} \tag{5}$$

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 $\mathcal{G}^{(1)}_{r_0=k}$ and the last order of the ending core tensor $\mathcal{G}^{(d)}_{r_d=k}$, can be reduced to matrices. Thus, along each of the R slices of $\mathcal{G}^{(1)}$, 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 $\mathcal{G}^{(1)}_{k,l_1} \mathcal{G}^{(2)}_{l_2} \cdots \mathcal{G}^{(d)}_{l_d,k}$ has the form tensor train decomposition. Therefore, the tensor ring model is essentially the linear combination of R different tensor train models. Figure 4 demonstrates the tensor ring structure, and the alternative interpretation as a summation of multiple tensor train structures.

5.2 TR-RNN model

5.2.1 Tensorizing $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{W}$

Without loss of generality, we tensorize the input vector $\mathbf{x}$, output vector $\mathbf{y}$, and weight matrix $\mathbf{W}$ into tensors $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{W}$, shown in Equation (6):

$$\begin{aligned} \mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times, \ldots, \times I_n}, \mathcal{Y} \in \mathbb{R}^{O_1 \times O_2 \times, \ldots, \times O_m} \\ \mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times, \ldots, \times I_n \times O_1 \times O_2, \ldots, \times O_m} \end{aligned} \tag{6}$$

where

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

5.2.2 Decomposing $\mathbf{W}$

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 (7). Without loss of generality, the core tensors corresponding to the input dimensions and output dimensions are grouped respectively, as shown in Figure 1.

$$\begin{aligned} TRD(\mathcal{W})_{i_1,\ldots,i_n,o_1,\ldots,o_m} = \sum_{r_0,\ldots,r_n,r_{n+1},\ldots,r_{n+m-1}} \\ \mathcal{G}^{(1)}_{r_0,i_1,r_1} \ldots \mathcal{G}^{(n)}_{r_{n-1},i_n,r_n} \mathcal{G}^{(n+1)}_{r_n,o_1,r_{n+1}} \ldots \mathcal{G}^{(n+m)}_{r_{n+m-1},o_m,r_0} \end{aligned} \tag{7}$$

The tensor contraction from input to hidden layer in TR form is shown in Equation (8).

$$\mathcal{Y}_{o_1,o_2,\ldots,o_m} = \sum_{i_1,\ldots,i_n} TRD(\mathcal{W})_{i_1,\ldots,i_n,o_1,\ldots,o_m} \mathcal{X}_{i_1,\ldots,i_n} \tag{8}$$

Compared with the redundant input-to-hidden weight matrix, the compression ratio in TR form is shown in Equation (9).

$$C_{TRD} = \frac{\prod_{i=1}^{i=n} I_i \prod_{j=1}^{j=m} O_j}{\sum_{i=1}^{n} R_{i-1} I_i R_i + \sum_{j=1}^{m} R_{n+j-1} O_j R_{n+j}} \tag{9}$$

5.2.3 Complexity Analysis

Since the weight tensor has been decomposed into the form of TRD with $n+m$ core tensors, the order of multiplication among input tensor and core tensors in Equation (8) determines the computational cost. In our implementation, we merge the input tensor with input core tensors and output core tensors sequentially. We can rewrite the Equation (8) as:

$$\begin{aligned} \mathcal{Y} = & \mathcal{X} \times_2^1 \mathcal{G}^{(1)} \times_{2,1}^{1,n+1} \mathcal{G}^{(2)} \ldots \times_{2,1}^{1,n-N+3} \mathcal{G}^{(N)} \ldots \\ & \times_{2,1}^{1,3} \mathcal{G}^{(n)} \times_1^2 \mathcal{G}^{(n+1)} \ldots \times_1^{M+1} \mathcal{G}^{(n+M)} \ldots \\ & \times_1^m \mathcal{G}^{(n+m-1)} \times_{3,1}^{1,m+1} \mathcal{G}^{(n+m)} \end{aligned} \tag{10}$$

In Equation (10), the symbols $\times_b^a$ and $\times_{\alpha,\beta,\theta\ldots}^{a,b,c\ldots}$ mean the Tensor Contraction Operation described in [5], a fundamental and the most important operation in tensor networks, and can be considered as a higher-dimensional analogue of matrix multiplication, inner product, and outer product. Note the N'th component can be formulated as $\times_{2,1}^{1,n-N+3} \mathcal{G}^{(N)}, N \in \{2, 3, \ldots, n\}$ and the $(n+M)$'th component can be formulated as $\times_1^{M+1} \mathcal{G}^{(n+M)}, M \in \{1, 2, \ldots, m-1\}$.

Our model is trained via back propagation. In Equation (10), according to the left-to-right multiplication order, our forward computational complexity could reach to $\mathcal{O}(nIR^3 + mOR^3)$ while the forward space complexity is $\mathcal{O}(IR^2)$, where all the ranks in our model are set in the same R. Comparison with some other compressing methods is shown in Table 4.

Table 4: Comparison among vanilla RNN, TT-RNN [24] and BT-RNN [25] and our model TR-RNN on forward complexity and memory usage. TT-RNN, BT-RNN and TR-RNN are all set in same rank R and $O_{max} = max_k(O_k), k \in \{1, 2, \ldots, d\}$. The d means the number of factors in BT-RNN and it is the number of cores in TT-RNN.

Method	Time	Memory
RNN	$\mathfrak{O}(IO)$	$\mathfrak{O}(IO)$
TT-RNN	$\mathfrak{O}(dIR^2O_{max})$	$\mathfrak{O}(RI)$
BT-RNN	$\mathfrak{O}(NdIR^dO_{max})$	$\mathfrak{O}(R^dI)$
TR-RNN	$\mathfrak{O}(nIR^3 + mOR^3)$	$\mathfrak{O}(R^2I)$

5.3 Details on the UCF11 Dataset

Through the real-world action recognition datasets UCF11(YouTube action dataset) [15], 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.

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.

End-to-End Training Recent years, some tensor decomposition models are proposed to classify videos like TT-LSTM [24], BT-LSTM [25] 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 sample. 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. In the experiment, the hyper-parameters in TT-LSTM and TR-LSTM models are set equally in their papers.

Pre-train with CNN Recently, some methods based on RNNs achieved higher accuracy by using the extracted feature as input vectors in computer vision [6]. 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.