Block-term Tensor Neural Networks

Jinmian Ye⁠,∗, Guangxi Li⁠,∗, Di Chen², Haiqin Yang³, Shandian Zhe⁴, Zenglin Xu⁠,†,∗∗

¹SMILE Lab, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610031, China.
²Center for Quantum Software and Information, University of Technology Sydney, NSW 2007, Australia
³Ping An Life Insurance Company of China, Ltd.
⁴Department of Computer Science, University of Utah, Salt Lake City, 84112 Utah, USA
⁵School of Computer Science and Engineering, Harbin Institute of Technology, Shenzhen, Shenzhen, 510085 Guangdong, China
⁶Center for Artificial Intelligence, Peng Cheng Laboratory, Shenzhen, 510085 Guangdong, China

Abstract

Deep neural networks (DNNs) have achieved outstanding performance in a wide range of applications, e.g., image classification, natural language processing, etc. Despite the good performance, the huge number of parameters in DNNs brings challenges to efficient training of DNNs and also their deployment in low-end devices with limited computing resources. In this paper, we explore the correlations in the weight matrices, and approximate the weight matrices with the low-rank block-term tensors. We name the new corresponding structure as block-term tensor layers (BT-layers), which can be easily adapted to neural network models, such as CNNs and RNNs. In particular, the inputs and the outputs in BT-layers are reshaped into low-dimensional high-order tensors with a similar or improved representation power. Sufficient experiments have demonstrated that BT-layers in CNNs and RNNs can achieve a very large compression ratio on the number of parameters while preserving or improving the representation power of the original DNNs.

Keywords: Tensor Networks, Network Compression, Neural Networks, Deep Learning

1. Introduction

Deep neural networks (DNNs) have achieved significantly improved performance in a number of applications, such as image classification, image captioning, video classification, speech recognition, machine translation, etc.¹ Represenative DNN architectures include Convolutional Neural Networks (CNNs)² and Recurrent Neural Networks (RNNs)³,⁴,⁵, which are used in capturing the spatial and temporal information from input data, respectively. However, due to their complex structure, the multiple layers and the huge amount of parameters, the training of DNNs becomes more difficult usually with higher space costs and more computational than classical machine learning models. For example, the winning network in the 2012 ImageNet Challenge by Krizhevsky et al. contains 60 million parameters with five convolutional layers and three fully-connected layers.⁶ As a result, the training of DNNs usually costs several days even on powerful Graphics Processing Units (GPUs), and the trained model also requires a huge memory consumption.⁷,⁸ Meanwhile, the redundancy of parameters in DNNs brings obstacles to the training procedure, probably leading to suboptimal local optimum. Therefore, compressing DNN architectures to reduce the parameter size is becoming an important issue in decreasing temporal and spatial complexity and reducing redundancy⁹,10,11.

To obtain compact neural networks, researchers have studied efficient low-rank approximation techniques in network design, since local correlations naturally exist in natural images²,¹²,¹³, and the dense connection is significantly inefficient at finding the spatial latent local correlation. Among the low-rank methods, tensor representations have demonstrated good performance in reducing the parameter sizes¹³,†⁴,†⁵,⁶. Some research works focus on speeding up the convolutional layers of DNNs¹⁷,¹⁶, and some on reducing the redundancy of the fully-connected layers (FC-layers) with tensor layers¹⁸,†⁴.

In this paper, we focus on reducing the huge parameter size of the fully-connected layers in DNNs. Therefore, we resort to the representation power of Block-term Tensor decomposition (BTD)¹⁸, and reduce the fully-connected layers in DNNs as the form of Block-term Tensor layers. Since the block-term tensor decomposition is a low-rank decomposition method combining Candecomp/Parafac (CP) decomposition¹⁹ and Tucker decomposition²⁰, it approximates a high-order tensor by the sum of Tucker models, which is more robust than the CP decomposition and the original Tucker decomposition. In addition, compared with the tensor train decomposition²¹, which often suffer from the instability of ranks²², we have demonstrated that Block-term decomposition can be more stable.

In detail, we transform the input data x into a tensor of various orders, and then replace the weight matrices (i.e., W) with new weights organized as a block-term tensor format (as will be shown in Figure 6). While in the training phase, the sparse-connected BT layer will learn the local correlations from the input. By integrating the new BT layer into CNNs and RNNs...
architectures, we obtain new compact neural network architectures, shorted as BT-CNN and BT-RNN respectively. An illustration can be found in Fig. 2 and Fig. 8 for CNN and RNN (in particular, LSTM), respectively. Compared with their vanilla versions, the new model can have a similar representation power while with much fewer parameters.

This paper extends the Block-term layer in LSTM [15] to more general neural network architectures. The major contributions of the proposed BT layer can be concluded as follows:

- The redesigned BT layer can reduce model parameters while retain the design philosophy of the original architecture. For LSTMs, the parameter reduction also speeds up the convergence, as demonstrated in Fig. 14.
- With the core tensor in the block term tensor structure, factors of the input data can have more interactions with each other, enhancing the ability to capture sufficient local correlations. Empirical results show that, compared with the standard neural network and the Tensor Train (TT) model [23], the BT model has a better expressive ability with the same amount of model parameters, as demonstrated in Tables 3 and 9.

This paper is organized as follows. We first introduce the tensor diagrams for representing block-term tensor decomposition, followed by the corresponding BT-CNN and BT-RNN models in Section 2. We then present evaluations of these models in Section 4 and Section 5, respectively. After reviewing and discussing recent work on tensor-based network compression methods in Section 6, we conclude our paper in Section 7.

2. Tensorizing Matrix Product

The core concept of this work is to approximate the matrix vector product $W \cdot x$ with much fewer parameters, while still preserving the representation ability in neural networks. The technique we use for the approximation is the low-rank Block Term Decomposition (BTD), which represents $W \cdot x$ as a series of light-weighted small tensor products.

2.1. Preliminaries and Background

2.1.1. Tensor Representation

A tensor in neural network, also known as a multi-way array, can be viewed as a higher-order extension of a vector (i.e., an order-1 tensor) and a matrix (i.e., an order-2 tensor). Like rows and columns in a matrix, an order-$d$ tensor $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ has $d$ modes whose lengths are represented by $I_1$ to $I_d$, respectively. The basic operations of tensors include linear arithmetic, tensor product, tensor transpose and tensor contraction. When the amount of tensors is big and the contraction relationships among their indices are complicated, a better way to represent them is using diagrams, namely tensor network diagrams. The basic symbols for tensor network diagrams are shown in Fig. 1 in which tensors are denoted graphically by nodes and edges. Each edge emerged from a node denotes a mode (or order, index) [24]. Here, we use boldface lowercase letters (e.g., $v$) to denote vectors, boldface capital letters (e.g., $W$) to denote matrices, and boldface Euler script letters (e.g., $A$) to denote higher-order tensors (order-3 or higher), respectively.

![Graphical representation](image)

![Symbol representation](image)

Figure 1: Two representations of vector $v \in \mathbb{R}^I$, matrix $M \in \mathbb{R}^{I \times J}$ and tensor $\mathbf{T} \in \mathbb{R}^{I \times J \times K}$.

to denote vectors, boldface capital letters (e.g., $W$) to denote matrices, and boldface Euler script letters (e.g., $A$) to denote higher-order tensors (order-3 or higher), respectively.

2.1.2. Tensor Contraction

Tensor multiplication is also known as tensor contraction. Tensor contraction between two tensors means that they are contracted into one tensor along the associated pairs of indices as illustrated in Fig. 2. Since a tensor is the high dimensional extension of a matrix or vector, we can refer the tensor contraction with matrix multiplication. Giving two matrices $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times J_2}$, we can conduct the multiplication if $I_2 = J_1$ as shown in Eq. 1.

$$
C_{i_1,j_2} = \sum_{k=1}^{l_2} A_{i_1,k} B_{k,j_2} 
$$

(1)

The matrix or vector multiplication is essentially the sum reduction along the same dimension. Extending this idea to high dimensional situations, we can obtain the tensor multiplication definition. For example, giving a 3-order tensor $\mathbf{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$, we can conduct the tensor multiplication as follows if $I_2 = J_1$:

$$
C_{i_1,j_3,j_2} = \sum_{k=1}^{l_2} A_{i_1,k,j_3} B_{k,j_2} 
$$

(2)

where the resulted tensor $C \in \mathbb{R}^{I_1 \times I_2 \times J_1 \times J_3}$ is a 4-order tensor. We use $\bullet_k$ to denote that the tensor multiplication is conducted along the $k$-th dimension.

We can also conduct multiplication along multiple dimensions. For example, if $I_2 = J_2$ and $I_3 = J_3$, we can reduce this two dimensions as:

$$
C_{i_1,j_1} = \sum_{k_1=1}^{l_2} \sum_{k_2=1}^{l_3} A_{i_1,k_1,k_2} B_{j_1,k_1,k_2} 
$$

(3)

Fig. 2 demonstrates the matrix multiplication and tensor contraction operation in the 3-order case.

Tensor contractions among multiple tensors can be computed by performing tensor contraction between two tensors many times. Hence, the order (i.e., the total number of modes) of an entire tensor network is given by the number of dangling (free) edges that is not contracted.
2.1.3. Block Term Decomposition (BTD)

There are usually two basic tensor decomposition methods: the CANDECOMP/PARAFAC decomposition (CP) [19] and the Tucker decomposition [20]. The CP method decomposes a tensor into a sum of several component rank-1 tensors; The Tucker method decomposes a tensor into a core tensor multiplied by a matrix along each mode. Their concentrations are different, as CP imposed a diagonal constraint on the core tensor of Tucker. Thus, a more general decomposition called Block Term (BT) decomposition, which combines CP and Tucker, has been proposed to take advantages of both of them [18].

The BT decomposition aims to decompose a tensor into a sum of several Tucker models with a low Tucker-rank. Specifically speaking, giving an order-\(d\) tensor \(\mathbf{X} \in \mathbb{R}^{I_1 \times \cdots \times I_d}\), its BT decomposition can be represented by multiple Tucker models. Fig. 3 illustrates an order-3 case. In the figure, \(G_n \in \mathbb{R}^{R_1 \times \cdots \times R_d}, n \in [1, N]\) denotes the order-\(d\) core tensors of the \(n\)-th Tucker model and each \(A_{n}^{(k)} \in \mathbb{R}^{I_k \times R_k}, k \in [1, d]\) denotes the \(k\)-th factor matrix in the \(n\)-th Tucker model. Moreover, \(\mathbf{X}\) can be computed as follows:

\[
\mathbf{X} = \sum_{n=1}^{N} G_n \cdot A_{n}^{(1)} \cdot A_{n}^{(2)} \cdot \cdots \cdot A_{n}^{(d)},
\]

where \(N\) denotes the CP-rank here, and \(R_k\) denotes the Tucker-rank while the \(d\) is the Core-order of the Tucker model.

2.2. Approximating Fully Connected Layer

This section demonstrates how to use the BTD to approximate fully connected layer, e.g., \(\mathbf{y} = \mathbf{W} \cdot \mathbf{x}\), to reduce model parameters. 1) The weight \(\mathbf{W}\) and the input vector \(\mathbf{x}\) should be tensorized to obtain the high order tensors \(\mathbf{W}\) and \(\mathbf{X}\); 2) then the weight tensor \(\mathbf{W}\) will be reconstructed by the BTD; 3) after that, the matrix multiplication \(\mathbf{W} \cdot \mathbf{x}\) can be replaced by a tensor contraction that \(\text{BTD}(\mathbf{W}) \bullet \mathbf{X}\); 4) the reconstructed model still uses the BP algorithm [25] to solve.

2.2.1. Step 1: Tensorization

To obtain the high order tensor structure, first we should tensorize the related tensors in fully-connected layer. Tensorization is an operation that transforms a vector or matrix into a high order tensor. As illustrated in Fig. 4, given a vector \(\mathbf{x} \in \mathbb{R}^{I_1 \times I_2 \times I_3}\), we can tensorize as an order-3 tensor \(\mathbf{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}\), essentially the reshape operation, also known as tensor folding.

2.2.2. Step 2: Applying BTD

Next we can decompose the weight tensor by the BTD algorithm as:

\[
\text{BTD}(\mathbf{W}) = \sum_{n=1}^{N} G_n \cdot A_{n}^{(1)} \cdot A_{n}^{(2)} \cdot \cdots \cdot A_{n}^{(d)},
\]

here the core tensor \(G_n \in \mathbb{R}^{R_1 \times \cdots \times R_d}\) has \(d\) dimensions and the \(R_1, \ldots, R_d\) denotes the Tucker-ranks. To avoid imbalance weight sharing, in this paper we set \(R = R_1 = R_2 = \cdots = R_d\). The factor tensor \(A_{n}^{(d)} \in \mathbb{R}^{I_d \times J_d \times R_d}\) is corresponding to the input and core tensor. To get the low-rank decomposition, we have the subjection of \(R_k \leq I_k\) (and \(J_k\)), \(k = 1, \ldots, d\).

2.2.3. Step 3: Substitution

After tensorizing the input vector and weight matrix, and decomposing the weight tensor, now we can conduct a tensor contraction operation between input tensor and BTD model as follows:

\[
\phi(\mathbf{W}, \mathbf{x}) = \text{BTD}(\mathbf{W}) \bullet \mathbf{X},
\]

\(\bullet_{1,2,\ldots,d}\) denotes that the contraction will be conducted along all \(d\) dimensions. To distinguish the new operation from the fully-connected layer, we call the proposed layer as the Block Term Layer (BTL). Fig. 6 demonstrates the substitution intuitively.

2.2.4. Step 4: Training BTL

In deep neural networks, the most common training method is the Back Propagation (BP) algorithm which back propagates the error in an reverse order with the forward computation and updates the learning parameters iteratively.

The BTL training still follows the standard error back-propagation algorithm, here we derive the weight gradient and input gradi-
When dealing with high dimensional input data, the fully connected layer between input and hidden state is extremely complex and requires a large amount of parameters. To reduce the complexity and the number of parameters, we replace the fully-connected layers with block-term layers (BT) in both Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

### 3. BT-Layer Implementations in CNNs and RNNs

#### 3.1. BT-CNN

In classical Convolutional Neural Networks, input features from stacked convolution layers are fed into fully-connected layers and activation layers, which serve as a non-linear classifier. To verify the efficient performance of the block-term layer, we replace the fully-connected layers with block-term layers in CNNs to construct a low-rank classifier.

As illustrated in Fig. 7, we only replace the redundant fully connected layers while keeping the other layers. We will discuss the compression ratio in the experiment section.

#### 3.2. BT-LSTM

We demonstrate the BT-Layer in LSTM since it is the most common used RNNs. Although our discussion mainly focuses on LSTM, the proposed method can easily be generalized into other RNN variants such as Gated Recurrent Unit (GRU) by following above procedures.

When dealing with high dimensional input data, the fully connected layer between input and hidden state is extremely complex and requires a large amount of parameters. To reduce the complexity and the number of parameters, we replace the fully-connected layers with block-term layers (BT) in both Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs).

The concept of Tucker-rank $R$ is similar to the rank of Singular Value Decomposition (SVD), which is the dimension of the core tensor. From Eq. (13), the #Params will increase exponentially with the increase of $R$. To obtain a low-rank model, we have $R \leq I_k$, $k \in [1, d]$. While the $I_k \approx \sqrt{dI}$, we have
In Eq. (15), the computation raised at the last contraction is \( O(J_{max} R^d) \). Since we have \( J_{max} R^d \leq J \) and \( R \leq J_{min} \), the complexity in the new computation order will be always smaller than the previous. Finally, we reduce the forward computation complexity from \( O(N d I J R) \) to \( O(N d I J_{max} R^d) \), which is more endurable.

We derive the gradients in backward phase and ignores the dimension alignment for simplicity:

\[
\frac{\partial L}{\partial A_n^{(k)}} = \frac{\partial L}{\partial Y} \cdot Y_t \cdot 1 \cdot A_n^{(1)} \cdot 2 \ldots
\]

\[
\cdots k - 1 \cdot A_n^{(k-1)} \cdot k + 1 \cdot A_n^{(k+1)} \cdot k + 2 \ldots
\]

\[
\cdots d \cdot A_n^{(d)} \cdot 1, 2, \ldots, d \cdot S_n,
\]

\[
\frac{\partial L}{\partial S_n} = X_t \cdot 1 \cdot A_n^{(1)} \cdot 2 \ldots d \cdot A_n^{(d)} \cdot 1, 2, \ldots, d \cdot \frac{\partial L}{\partial Y}
\]

Both Eq. (16) and Eq. (17) follow the similar computation pattern with Eq. (6), raising the same complexity in \( O(d I J_{max} R^d) \).

The detail of the computation complexity in various architectures is shown in Table 1.

### 4. Evaluation on BT-CNN

The goal of this experiment is to verify the effectiveness of our proposed BT-nets using various neural network architectures in terms of the compression ratio and accuracy of BT-layers. For fair comparison, we adopt pre-fixed architectures of CNNs available in various platforms (e.g., Tensorflow and MxNet) as our test beds, and replace only the FC-layers with BT-layers. As tensor train methods have achieved better performance than SVD in compressing CNNs (as shown in [13]), we focus to compare our proposed method with tensor train methods.

For consistency, we adopt three three widely-used benchmark datasets, i.e., MNIST [26], Cifar10 [27], and ImageNet [28].

The experiments are performed on a server with one NVIDIA TITAN Xp GPU. For fair comparison, we train all networks from scratch with stochastic gradient descent with momentum of 0.9. To avoid the influence of random initialization and the problem of gradient vanishing or exploding, it is necessary to add a batch normalization layer after the BT-layer.

### 4.1. Results on MNIST

As a baseline we use the convolutional neural network LeNet-5 [29], with two convolutional (plus activation function and
max-pooling) layers followed by two fully-connected layers (FC-layers) of size $800 \times 500$ and $500 \times 10$.

We fix the convolutional part of the original network and only replace the first FC-layer with the BT-layer. The BT-layer reshapes the input and output tensors as $5 \times 5 \times 8 \times 4$ and $5 \times 5 \times 5 \times 4$ tensors respectively. As the prediction task for MNIST is quite easy, we simply use one block term decomposition (which is Tucker decomposition and $N = 1$) and vary the Tucker-rank from 2 to 3 ($R = 2$ or $R = 3$). Here we use $BT - N : 1 - R : 2$ to denote a BT network with the CP rank equal to 1 and the Tucker-rank equal to 2, which is also shorted as $BT - R : 2$ when $N = 1$. We run the TT-net\(^1\) as a competitor by replacing the first FC-layer with the TT-layer. For a fair comparison with a similar amount of parameters, we set the TT-rank to 2, and the corresponding network is denoted by $TT - R : 2$.

Table 2 reports the results on the MNIST dataset. It can be observed that the parameter amount is greatly reduced while with a slight improvement in accuracy. The first column represents the different network architectures, the middle two columns represent the number of parameters in the first FC-layer (or its alternatives) and the compression ratio respectively. The last column represents the accuracy on the test set. We can see at first glance that the number of parameters in the FC-layer can be reduced from $800 \times 500$ to 228 in “BT-R:2” and the compression ratio can up to 1754, with almost the same accuracy. The compression ratio of the entire network is about 14. We can also observe that “BT-R:3”, with 399 parameters in BT-layer, has the same accuracy as the baseline while TT-net lost 0.03% in performance on about the same order of magnitude of the parameter amount.

### 4.2. Results on CIFAR10

We refer to the tensorflow implantation\(^2\) as the baseline CNN implementation, which consists of two Convolutional, Local Respond Normalization (LRN) and Max-pooling layers followed by three FC-layers of size $2304 \times 384$, $384 \times 192$ and $192 \times 10$, respectively.

We similarly replace the first FC-layer with BT-layer which reshapes the input and output dimensions as $6 \times 6 \times 8 \times 8$ and $6 \times 4 \times 4 \times 4$ respectively. TT-net replaces the first FC-layer with TT-layer which has the same output dimension as BT-layer. For fair comparison, we let the CP-rank vary from 1 to 8 and the Tucker-rank vary from 1 to 3 in the BT-layer, and let the TT-rank equal 2 and 8 in the TT-layer.

Some results of the CIFAR10 dataset are reported in Table 3 and others can be found in Fig. 10. We can see that when using the “BT-N:1-R:2” structure, the compression ratio is up to 3351 at the cost of about 1% reduction in accuracy. By comparison, the compression ratio of “TT-R:2” is only 2457 with almost the same accuracy as the BT-layer. In response to the increase in the architecture’s complexity, we observe that “BT-N:4-R:3” has a larger compression ratio while obtaining a better accuracy at the same time compared with “TT-R:8”. “BT-N:4-R:3” can have a total compression ratio of 2.98.

### 4.3. Results on ImageNet

The ILSVRC 2012 (ImageNet) is a large dataset which consists of 1.2 million images for training and 50,000 for validation and comprises 1000 classes. As a baseline, we use the AlexNet architecture\(^3\) which has three FC-layers of size $6400 \times 4096$, $4096 \times 4096$ and $4096 \times 1000$.

We replace the first FC-layer with BT-layer where the input and output dimensions are reshaped as $10 \times 10 \times 8 \times 8$ and $8 \times 8 \times 8 \times 8$ respectively. The same dimension reshaping is performed in TT-layer as well. As in the CIFAR10 case, we experiment with two groups of variations (simple and complex) of the BT-layer and the TT-layer. Accordingly, we choose “BT-N:1-R:2” and “BT-N:4-R:2” as BT-layers and set TT-rank as 2 and 8 in TT-layers.

In Table 4 we report the compression ratio, Top-1 and Top-5 accuracy on different architectures. From the results we see that BT-layer in the best case (“BT-N:4-R:2”) can get a compression ratio of 11070 (from 6400 parameters to 2368) on amount of parameters while achieving a slightly better Top-1 and Top-5 accuracy than baseline at the same time. The total compression ratio of the entire network is 2.06. By comparison, “TT-R:8” only gets a compression ratio of 2528 and even about 1% accuracy drop. Similarly, “BT-N:1-R:2” gets a compression factor of more than 40,000 with 2.2% decrease in Top-5 accuracy, better than “TT2”. Please note that all the experiments on the Imagenet are performed without fine-tuning.

### 4.4. Extended Experiment

#### 4.4.1. Extension to Convolutional Layers

The weight in a convolutional layer (conv-layer) is usually denoted by an order-4 tensor $W \in \mathbb{R}^{H \times W \times C_{in} \times C_{out}}$, where

\(^1\)https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10

\(^2\)https://github.com/apache/incubator-mxnet/tree/master/example/image-classification

---

Table 2: Results on MNIST with various Tucker Rank settings. By substituting the first fully connected layer, we got great performance in both TT and BT networks.

| Architecture | # params | Comp.R | Acc (%) |
|--------------|----------|--------|---------|
| Baseline     | 800×500  | 1      | 99.17   |
| TT-R:2       | 342      | 1169   | 99.14   |
| BT-R:2       | 228      | 1754   | 99.14   |
| BT-R:3       | 399      | 1002   | 99.18   |

Table 3: Results on CIFAR10 with various architectures.

| Architecture | # params | Comp.R | Acc (%) |
|--------------|----------|--------|---------|
| Baseline     | 2304×384 | 1      | 85.99   |
| TT-R:2       | 360      | 214    | 84.90   |
| TT-R:8       | 4128     | 214    | 85.70   |
| BT-N:1-R:2   | 264      | 3351   | 84.95   |
| BT-N:4-R:2   | 1056     | 838    | 85.47   |
| BT-N:4-R:3   | 1812     | 488    | 85.83   |
Table 4: Results on ImageNet with various architectures.

| Architecture              | Comp.R | Top-1 Acc (%) | Top-5 Acc (%) |
|---------------------------|--------|---------------|---------------|
| Baseline                  | 1      | 56.17         | 79.62         |
| TT-R:2                    | 30340  | 52.14         | 76.40         |
| TT-R:8                    | 2528   | 55.11         | 78.61         |
| BT-N:1-R:2                | 44281  | 53.20         | 77.38         |
| BT-N:4-R:2                | 11070  | 56.48         | 79.69         |

Table 5: Results of adding FC, TT and BT layers to ResNet-20 on cifar10.

| Architecture | # Parameters | Acc (%) |
|--------------|--------------|---------|
| Baseline     | -            | 91.73   |
| add FC       | 1024 × 1024  | 90.69   |
| add TT2      | 480          | 88.97   |
| add TT8      | 5760         | 90.30   |
| add BT-N:1-R:2 | 336   | 90.58   |
| add BT-N:4-R:2 | 1344 | 92.13   |

H and W denotes the height and width of the perception area, and \(C_{in}\) and \(C_{out}\) denotes the size of input channels and output channels. Since the first three indices are representing the input and the last one represents the output, we can view the weight tensor \(W\) as a matrix \(W \in \mathbb{R}^{H \times W \times C_{in} \times C_{out}}\), then we perform the generalized BT decomposition on it as stated above. Hence, we could transform conv-layer to BT-layer as well.

However, due to the conv-layer’s specific characteristics, we should pay attention to the following:

- As the matrix \(W \in \mathbb{R}^{H \times W \times C_{in} \times C_{out}}\) is inherently belonging to a tensor, we require dividing the dimensions according to the original dimensions, for example \(H \times W \times C_{in}^{1} \times C_{in}^{2}\) and \(C_{out}^{1} \times C_{out}^{2}\), to maintain the high-order structure information of the convolutional layers.

- In general, \(H\) and \(W\) are far less than \(C_{in}\) and \(C_{out}\), so the key is dividing the larger \(C_{in}\) and \(C_{out}\) suitably. In practice, we divide the input and output dimensions as \(H \times W \times C_{in}^{1} \times C_{in}^{2} \cdots\) and \(1 \times 1 \times C_{out}^{1} \times C_{out}^{2} \cdots\), respectively.

4.4.2. Results of Extensions to Conv-layer

Since the CIFAR-Net\(^1\) has the largest parameter occupation rate of conv-layers among the above three networks, we evaluate the results of extensions to conv-layer on the CIFAR10 dataset. In CIFAR-Net, the weight in the second conv-layer, which contains the most parameters, which is \(5 \times 5 \times 64 \times 64\). We replace it with a BT-layer where the dimensions are reshaped as \(5 \times 5 \times 64\) and \(1 \times 1 \times 64\), respectively. We set the CP-rank as 2, Tucker-rank as 3, and the corresponding TT-rank as 6. For a better comparison, we optionally substitute the two layers. The results are illustrated in Table 6. We can see that “BT-N:2-R:3 + FC”, which only replaces the conv-layer with a BT-layer, obtains the best accuracy since BT gives a better representation of the weight of the conv-layer. In addition, “BT-N:2-R:3 + BT-N:2-R:3” could get a total Comp.R of 10.10 with only about 0.5% reduction in accuracy, while “TT-R:6 + TT-R:6” reduces about 1% in accuracy with a total Comp.R of 9.87.

4.4.3. Results of applying BT-layers to the U-Net structures

Since the U-Net is a relatively small-scaled convolutional neural network which includes multiple convolutional and de-convolutional layers, we evaluate the results of applying BT-layers to the U-Net. As a baseline, we use the U-net structure and conduct the detection task of radio frequency interference (RFI) in radio astronomy as the same as [30]. We replace the last conv-layer in the U-net with the BT-layer, because the last conv-layer contains the most parameters, which is \(3 \times 3 \times 256 \times 256\). The results are illustrated in Table[7] We can see that applying the BT-layer (“BT-N:1-R:2”) could get could get a compression ratio of 4.50 with only about 0.8% reduction in accuracy. BT-layer in the another case (“BT-N:2-R:3”) can get a compression ratio of 1.50 on amount of parameters while achieving slightly better accuracy than baseline.

4.4.4. Comparison between pyramid-structure CNN and BT-layers

As described in [31], the pyramid-structure CNN starts from a big first layer and then refines the features at each higher layer, until it achieves a reduced most discriminative set of features like a pyramid. We change the filter numbers of CIFAR-Net to construct a pyramid-structure CNN and replace the first FC-layer with the BT-layer. The results are illustrated in Table[8] As the same as [31], we use “C10: 64-64-384-192-10” to denote a CNN which has 64 filters in the first convolutional layer and 64 filters in the second convolutional layer followed by several fully-connected layers. “SPryC10” denotes a

---

1https://github.com/tensorflow/models/blob/master/research/slim/nets/cifar10.py
4.4.5. Sensitive Analysis of BT-ranks and the Tensor-order

As a BT-layer has two kinds of ranks, e.g., CP-rank \((N)\) and Tucker-rank \((R)\), and the core-order \(d\), we utilize the CIFAR10 dataset as an example to study their impacts on the performance. The network architectures are designed as follows: we just replace the first FC-layer with BT-layer where the CP-rank varies from 1 to 8, the Tucker-rank from 1 to 3 and the tensor-order \(N\) from 2 to 6.

The results about BT-ranks are reported in Fig. 10. We can intuitively see that higher test accuracy can be achieved with larger BT-ranks. In detail, we observe that when the number of blocks (CP-rank) is small, the accuracy curves rise quickly, but when the CP-rank becomes large, the curves are almost horizontal. Similar observations can be found in the Tucker-rank. Thus, if we want to get a better performance, we need to consider both and let them be appropriate values. The results about \(N\) are reported in Fig. 11. In general, higher accuracy can be obtained with a larger \(N\). But when \(N\) is too small, the accuracy does not go up because a smaller \(N\) cannot sufficiently capture the weight’s higher-order features. However, when \(N\) is large, the performance is not dissatisfied as well due to the too small dimensions. Hence, we choose \(N = 4\) in this paper.

4.4.6. Extension of adding BT-layers to ResNet

It is noted in [32] that the FC layer can play an important role in transferring knowledge, although there are no FC-layers in some recent structures, such as ResNet [33], due to the high computational complexity. Following [32], we test the performance of adding an FC-layer, a TT-layer and a BT-layer respectively. As a baseline, we use the ResNet-20 architecture\(^4\). The dimensions of FC-layer are set as \(1024 \times 1024\), same as [32], and we reshape them as \(8 \times 8 \times 4 \times 4\) and \(8 \times 8 \times 4 \times 4\) respectively in TT-layer and BT-layer. The results are illustrated in Table 5. We can see that adding an FC-layer reduces the accuracy by about 1%, while adding a BT-layer with moderate ranks (e.g., “BT-N:4-R:2”) which only has additional 1344 parameters, could bring 0.4% higher accuracy than the baseline. This sufficiently demonstrates that BT-layer could improve the network’s performance, probably due to its ability of connecting high level features with the labels.

4.4.7. Visualization

For explaining and interpreting the proposed architectures, we apply the Grad-CAM [34] visualization techniques to the proposed architectures. Taking the CIFAR10 dataset as an example, we visualize the last conv-layer of CNN and then replace it with an BT-layer (here “BT-N:2-R:3” is used). The result of visualization is shown in Fig. 12. We can see from the figure that the BT-layer focuses more on key areas, e.g. the head of that dog here.

---

\(^4\)https://github.com/tornadomeet/ResNet
4.4.8. Results of Running Time

In order to explore the time consumption of BT-layers, we test on a single $6400 \times 4096$ FC-layer and the corresponding BT-layers which have the same architectures as in the ImageNet experiment. These benchmark experiments are performed on TensorFlow with a single Tesla k40m GPU. The results of the training time and inference time with different batch sizes are showed in Fig. 13. From the results we can intuitively observe that when BT-ranks are small, BT-layer has a significant acceleration effect compared with FC-layer. When BT-ranks are increased to let the BT-Net catching the original performance, the time cost in BT-layer is also competitive with FC-layers.

![Figure 13: Running time of $6400 \times 4096$ FC-layer and its corresponding BT-layers.](image)

Table 9: Some important results reported in literature on the UCF11 dataset with the video classification task.

| Method                  | Accuracy |
|-------------------------|----------|
| Orthogonal Approaches   |          |
| Original [35]           | 0.712    |
| Spatial-temporal [36]   | 0.761    |
| Visual Attention [37]   | 0.850    |
| RNN Approaches          |          |
| LSTM                    | 0.697    |
| TT-LSTM [38]            | 0.796    |
| BT-LSTM                 | 0.853    |

5. Evaluation on BT-RNN

We present our experiments conducted in two different parts: 1) quantitative evaluation. We use a video classification task show that compares to vanilla LSTM and TT-LSTM, our model achieve better performance while consuming fewer parameters, since BT model can capture more spatial information in fewer parameters. 2) qualitative evaluation. An image generation task and an image caption task will be conducted to show that BT model can handle various input data and output high-quality result.

5.1. Evaluations of BT-LSTM on the Task of Video Classification

UCF11 YouTube Action [35] is a large-scale dataset used in video classification task, containing 1600 video clips with 11 action categories. A video clip can be seen as a sequence of image frames naturally. But it’s extremely difficult to train an RNN model to accomplish this task as the high dimension input data. Our data preprocessing follows the procedure reported by [38], which down-sampling from $320 \times 240 \times 3$ to $160 \times 120 \times 3$ and 6 frames will be random sampled for each video
To evaluate the performance of BT-LSTM against LSTM and TT-LSTM, we construct the classification model with a single RNN cell with various types. Fig. 14 demonstrates the performance of 3 architectures with various hyper-parameter settings. Since the RNN design philosophy is kept in our model, the other orthogonal approaches such as [37] and [38], will be equipped together to improve the accuracy.

The hyper-parameters of three architectures are set as follows: all the hidden size in RNN blocks are set as 256; all the Core-order of BT-LSTM models are set as \( d = 4 \); the input data are tensorized as \( I_1 = 8, I_2 = 20, I_3 = 20, I_4 = 18 \); while the rank of TT-LSTM is \( R_1 = R_5 = 1, R_2 = R_3 = R_4 = 4 \) and the Tucker-ranks in BT-LSTM models are various.

The training and validation results of BT-LSTM and the comparison methods are presented in Fig. 14. The number of parameters of the vanilla LSTM model is 58.9M, while the BT-LSTM with Tucker-rank 1,2,4, is 721, 1470, and 3387 respectively, raising a very high compression ratio. However, the BT-LSTM models still outperform the vanilla LSTM model with the highest 15.6%, saying that the BT-LSTM model is an effective way to reduce the model parameter while improving model accuracy. We also reproduce the TT-LSTM and find that with comparable parameters, BT-LSTM(R=4) outperforms TT-LSTM(R=4) with 7.2%.

With much less parameters, BT-LSTM raises a higher convergence speed than the vanilla LSTM. According to Fig. 14(b), BT-LSTM achieves the validation accuracy of 60% at the 16th epoch, while the vanilla LSTM needs about 220 epochs. TT-LSTM has nearly the same parameters with BT-LSTM, so as the convergence speed.

Table 9 gives the state-of-the-art results in UCF11 dataset. Comparing to other methods, our BT-LSTM provides a more elegant solution with sparse connection in LSTM architecture.

5.2. Evaluations of BT-LSTM on Image Captioning Task

Image Captioning is a challenging task that generates a readable sentence to describe an image. The base model we used is the Neural Image Captioning [39] and the dataset is the large-scale MSCOCO [40] dataset. MSCOCO contains 82783 images for training and 40775 images for testing. The model uses a pre-trained Inception-V3 model as the backbone architecture to extract the image features and a sentence-to-sentence LSTM to generate sentence. We also follow the data pre-processing described in [39] that scales images to 224 x 224 and subtracts the channel means.

In this model, the input data of BT model is heterogeneous vectors merged by the image feature vector generated by CNN backbone and word word feature vectors generated by embedding layers. According to Fig. 15 both LSTM and BT-LSTM can pronounce human-readable sentences, but the sentences generated by BT-LSTM is more close to the ground truth.

6. Related Work

Extensive research on DNNs has proven that the dense connection is significantly inefficient at extracting the spatially latent local structures and local correlations naturally exhibited in the data [2, 11].

Reducing the computation or storage complexity is the holy grail of the neural networks design. There have been a number of network slimming techniques motivated from different perspectives [9], including network hashing [41], network sparsification or pruning [42, 43, 44, 45, 46, 47], binarization and quantization [48, 49], low rank representation [50, 51, 52, 53, 54, 55, 56, 57], and knowledge distillation [58].

Among these methods, the low-rank methods is a type of attractive approach to implicitly prune connections, which employs matrix factorization or tensor factorization techniques [59, 60, 61, 62, 63, 64, 65] on the weighting matrices to obtain compact weight matrices or tensors. Some of the traditional tensor decomposition methods have been applied in CNNs or RNNs to alleviate the inefficient fully-connected structure [13, 38, 66, 53, 44], achieving higher accuracy or better parameter reduction than the traditional matrix decomposition based methods [67, 68, 69]. The accuracy improvement is mainly from the capture of the high-order spatial information. In addition, tensor decomposition can also be used in accelerating the convolution operation.

Recently, tensor networks [70, 71, 72] has attracted the attention of researchers in the machine learning area [73, 74, 75].
Recently, Novikov et al. applied the Tensor Train method to replace the fully-connected layer in CNNs, which can reduce a huge amount of parameters while speeding up the inference phase [13]. Tensor Train based structure also applied in RNNs to solve the high dimensional input data [39]. Other tensor network decomposition methods also applied in Deep Neural Networks (DNNs) for various purposes [17][26][14][17][78][79].

It is important to note that although recent leading CNNs architectures, e.g., DenseNet [80] and ResNet [33], try to cut out the huge fully connected layer, there are no theories to prove the necessity of avoiding fully connected layers. In addition, it has been shown that the fully connected layers have good knowledge transfer ability [32]. Furthermore, we show that a block-term tensor layer can be added into ResNet to further improve the performance, as demonstrated in Table 5.

### 7. Conclusion

We have presented a new network architecture design in which the commonly used fully-connected layers in CNNs and RNNs are replaced with the Block Term-layers. Thanks to the better representation ability of the block-term tucker structure, BT-layers can not only greatly reduce the number of parameter size, but also improve the generalization performance especially for recurrent neural networks. Our experiments on several datasets have demonstrated the promising performance of the BT-layers in both CNNs and RNNs.

As for future work, in order to obtain higher compression ratio for real applications, we plan to combine the BT-Nets with other compression techniques, such as pruning and binarization.

### Acknowledgement

This paper was partially Supported by the National Key Research and Development Program of China (No. 2018AA0100204). Dr. Haiqin Yang is also affiliated with Dept. of Decision Sciences and Managerial Economics, The Chinese University of Hong Kong, Hong Kong, and Dept. of Computing, The Hang Seng University of Hong Kong, Hong Kong. We thank Dr. Shuicheng Yan and Mr. Linnan Wang for valuable comments and discussions to this paper, and thank Dr. Xinqi Chu for providing computing resources.

### References

[1] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85 – 117.

[2] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks 3361 (10) (1995) 1995.

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

[4] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Advances in neural information processing systems, 2014, pp. 3104–3112.

[5] H. Liu, L. He, H. Bai, B. Dai, K. Bai, Z. Xu, Structured inference for recurrent hidden semi-markov model, in: IJCAI, ijcai.org, 2018, pp. 2447–2453.

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[7] L. Wang, Y. Yang, R. Min, S. Chakrabhuri, Accelerating deep neural network training with inconsistent stochastic gradient descent, Neural Networks 93 (2017) 219 – 229.

[8] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, T. Kraska, Super-neurons: Dynamic gpu memory management for training deep neural networks, in: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '18, 2018, pp. 41–53.

[9] Y. Cheng, D. Wang, P. Zhou, T. Zhang, A survey of model compression and acceleration for deep neural networks (2017). [arXiv:1710.09282](https://arxiv.org/abs/1710.09282)

[10] Z. Lu, V. Sindhwani, T. N. Sainath, Learning compact recurrent neural networks, in: Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on, IEEE, 2016, pp. 5960–5964.

[11] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

[12] Y. Cheng, D. Wang, P. Zhou, T. Zhang, A survey of model compression and acceleration for deep neural networks., CoRR abs/1710.09282.

[13] A. Novikov, D. Podoprikhin, A. Osokin, D. P. Vetrov, Tensorizing neural networks, in: Advances in Neural Information Processing Systems, 2015, pp. 442–450.

[14] J. Kossaifi, Z. C. Lipton, A. Khanna, T. Furlanello, A. Anandkumar, Tensor regression networks, arXiv preprint arXiv:1707.08308.

[15] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, Z. Xu, Learning compact recurrent neural networks with block-term tensor decomposition, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[16] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS’14, MIT Press, Cambridge, MA, USA, 2014, pp. 1269–1277.

[17] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky, Speeding-up convolutional neural networks using fine-tuned cp-decomposition, arXiv preprint arXiv:1412.6553.

[18] L. De Lathauwer, Decompositions of a higher-order tensor in block terms—part i: Definitions and uniqueness, SIAM Journal on Matrix Analysis and Applications 30 (3) (2008) 1033–1066.

[19] J. D. Carroll, J.-J. Chang, Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition, Psychometrika 35 (3) (1970) 283–319.

[20] L. R. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (3) (1966) 279–311.

[21] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Annals of Physics 349 (2014) 117–158.

[22] Q. Zhao, G. Zhou, S. Xie, L. Zhang, A. Cichocki, Tensor ring decomposition, arXiv preprint arXiv:1606.05355.

[23] I. V. Oseledets, Tensor-train decomposition, SIAM Journal on Scientific Computing 33 (5) (2011) 2295–2317.

[24] A. Cichocki, Era of big data processing: A new approach via tensor networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2381–2389.

[25] P. J. Werbos, Backpropagation through time: what it does and how to do it, Proceedings of the IEEE 78 (10) (1990) 1550–1560.

[26] Y. LeCun, The mnist database of handwritten digits, http://yann.lecun. com/exdb/mnist/.

[27] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images.

[28] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, L. Fei-Fei, Imagenet large scale visual recognition competition 2012 (lsvrc2012) (2012).

[29] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[30] J. Akeret, C. Chang, A. Lucchi, A. Refregier, Radio frequency interference mitigation using deep convolutional neural networks, Astronomy and Computing 18 (2017) 35–39.
eralization of Neural Networks and Application to Model Compression, arXiv e-prints (2018) arXiv:1805.10352.

[79] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, D. Song, M. Zhou, A tensorized transformer for language modeling, CoRR abs/1906.09777.

[80] G. Huang, Z. Liu, K. Q. Weinberger, L. van der Maaten, Densely connected convolutional networks, arXiv preprint arXiv:1608.06993.