Journal of Systems Engineering and Electronics
Vol. PP, No. 99, November 2023, pp.1 – 23

Low rank optimization for efficient deep learning: making a balance between compact architecture and fast training

OU Xinwei, CHEN Zhangxin*, ZHU Ce, and LIU Yipeng*

School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China

Abstract: Deep neural networks (DNNs) have achieved great success in many data processing applications. However, high computational complexity and storage cost make deep learning difficult to be used on resource-constrained devices, and it is not environmental-friendly with much power cost. In this paper, we focus on low-rank optimization for efficient deep learning techniques. In the space domain, DNNs are compressed by low rank approximation of the network parameters, which directly reduces the storage requirement with a smaller number of network parameters. In the time domain, the network parameters can be trained in a few subspaces, which enables efficient training for fast convergence. The model compression in the spatial domain is summarized into three categories as pre-train, pre-set, and compression-aware methods, respectively. With a series of integrable techniques discussed, such as sparse pruning, quantization, and entropy coding, we can ensemble them in an integration framework with lower computational complexity and storage. In addition to summary of recent technical advances, we have two findings for motivating future works. One is that the effective rank, derived from the Shannon entropy of the normalized singular values, outperforms other conventional sparse measures such as the ℓ_1 norm for network compression. The other is a spatial and temporal balance for tensorized neural networks. For accelerating the training of tensorized neural networks, it is crucial to leverage redundancy for both model compression and subspace training.

Keywords: model compression, subspace training, effective rank, low rank tensor optimization, efficient deep learning.

DOI: 10.23919/JSEE.2023.000159

1. Introduction

Deep neural networks (DNNs) have been widely used in many data processing applications, such as speech recognition, computer vision [1−4], natural language processing [5,6], etc. As a deeper or wider structure can lead to better performance, DNNs are gradually characterized by their over-parameterization. Over-parameterization, on the other hand, suggests too much redundancy in DNNs, which leads to overfitting [7,8]. There are mainly two challenges in deep learning: high complexity and slow convergence. High complexity means that there are millions of parameters in DNNs, and computation between massive parameters and inputs is cumbersome, which underlines the need for efficient algorithms to compress and accelerate. For example, the number of parameters in Visual Geometry Group (VGG)-16 [2] is almost seven million. For an image in ImageNet dataset [1] with a size of 224×224×3, the feedforward process requires 30.9 billion float point-operations (FLOPs). The high complexity is unaffordable for resource-limited devices, such as mobile phones [9] and Internet of Things (IoT) devices [10]. The slow convergence is caused by the conventional back propagation (BP) algorithm, resulting in time-consuming training [11]. Also, the convergence speed is sensitive to the setting of the learning rate and the way to initialize weights.

There are many works attempting to reduce the high complexity of DNNs with acceptable performance decay. The investigation of model compression can be summarized into two folds: one is to reduce the number of parameters, and the other is to reduce the average bit width of data representation. The first fold includes but is not limited to low rank approximation [12−15], pruning [16,17], weight-sharing [18], sparsity [19], and knowledge distillation [20]. Since these techniques have their own limitations, it is better to combine them to fully exploit the redundancy in DNNs. Quantization [21,22] and entropy coding [20] belong to the second category, which is designed to achieve a lower number of bits per parameter.

Manuscript received September 21, 2022.

*Corresponding authors.

This work was supported by the National Natural Science Foundation of China (62171088, U19A2052, 62020106011), and the Medico-Engineering Cooperation Funds from University of Electronic Science and Technology of China (ZYGX2021YGLH215, ZYGX2022YGRH-005).

Low rank approximation has been widely adopted due to its strong theoretical basis and ease of implementation on hardware. In this survey, we comprehensively review this rapidly developing area by dividing low rank optimization for model compression into three main categories: pre-train method, pre-set method, and compression-aware method. The biggest distinction among them is the way to train. The pre-train method directly decomposes a pre-trained model to get warm initialization for the compressed format, followed by retraining the compressed model to recover the performance. Without pre-training, the pre-set method trains a network that is pre-set to a compact format from scratch. Totally different from the above two methods, the compression-aware method explicitly accounts for compression in the training process by gradually enforcing the network to enjoy low-rank structure. Although the discussion about low rank optimization can also be found in [23], we further investigated how to integrate it with other compression techniques to pursue lower complexity and recommended the effective rank as the most efficient measure used in low rank optimization.

When the redundancy in DNNs is exploited by subspace training, DNNs can converge faster without losing accuracy. In deep learning, it is conventional to train networks with first-order optimization methods, e.g. stochastic gradient descent (SGD) [24], which is computationally cheap. But there are some inherent drawbacks to first-order optimization methods, such as slow theoretical and empirical convergence. Second-order methods can deal with such a problem well, but because of the heavy computational burden of Hessian matrices, second-order methods are not applicable to DNNs. The idea that projecting parameters onto a tiny subspace represented by several independent variables is an effective way to solve this problem. Since only a few variables need to be optimized, we can apply second-order optimization methods to achieve the temporal efficiency of deep learning.

In this survey, we first present a comprehensive overview of various tensor decomposition methods applicable to model compression. Next, the low rank optimization for model compression is summarized in terms of pre-set, pre-train, and compression-aware methods. For each method, a detailed discussion on key points about implementation is given. More meticulously, we make a comparison among various sparsity measures used in the compression-aware method, and dig out the most efficient measure, i.e., effective rank, which is seldom used as a sparse regularizer before. In addition, while there are already many works that give a list of joint-way compression [25,26], little attention has been paid to the integration between low rank approximation and other compression techniques. Therefore, we present an overall survey on this kind of integration here. Then, we introduce low rank optimization for subspace training. Furthermore, we are the first to relate these two types of low rank optimization, discovering that redundancy in the temporal domain and spatial domain are of the same origin. And there is a discussion on how to apply subspace training on tensorized neural networks to achieve spatial efficiency and temporal efficiency simultaneously.

Different from the previous surveys on tensors for efficient deep learning [15,27,28], the main contributions of this paper can be summarized as follows.

(i) We make a detailed overview of low rank approximation for model compression, and we find that recurrent neural networks (RNNs) can be effectively compressed using hierarchical Tucker (HT) decomposition and Kronecker product decomposition (KPD), convolutional neural networks (CNNs) can be effectively compressed using tensor train (TT), and generalized Kronecker product decomposition (GKPD), while tensor ring (TR) and block term decomposition (BTD) can suitably compress both RNNs and CNNs.

(ii) A series of integratable neural network compression techniques are discussed in details, and an integration framework is summarized to well take advantage of various methods.

(iii) We analyse that the redundancy in the space domain and time domain are of the same origin. In order to accelerate the training of tensorized neural networks, we should make the balance between spatial efficiency and temporal efficiency.

(iv) After discussing and testing various sparse measures for low rank optimization for DNN compression, the effective rank outperforms in numerical experiments.

This survey is organized as follows. In Section 2, we give an overview of low rank optimization for model compression. Low rank approximation integrated with other compression techniques is reviewed in Section 3. Section 4 introduces low rank optimization for subspace training and analyses the coupling between these two types of low rank optimization.

2. Low rank optimization for model compression

We provide an overall mind map of low rank optimization in Fig. 1. In this section, we focus on the spatial efficiency. Since DNNs are over-parameterized, there are opportunities to make deep networks more compact.

Compression methods, like quantization, pruning, and low-rank approximation, can lower complexity of DNNs without significant accuracy degradation. Among them, low rank approximation has been widely adopted because of the solid theoretical basis of tensor decomposition [29]. In this section, we first introduce various tensor decomposition methods applicable for network compression, and then divide optimization methods for low rank approximation into three categories: pre-train, pre-set, and compression-aware methods. In addition, we make a discussion on efficient sparsity measures.

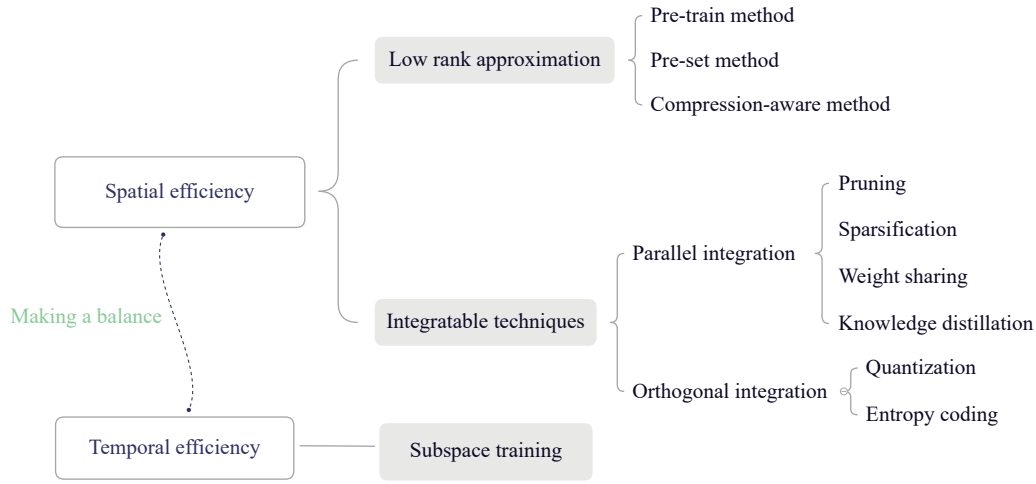

Fig. 1 Overview of low rank optimization for efficient deep learning

2.1 Tensor decomposition

Low rank approximation can provide an ultra-high compression ratio for RNNs with insignificant accuracy loss. However, when it comes to CNNs, the compression performance is not as satisfying as RNNs. In early literatures, four-dimensional (4D) convolutional kernels are reshaped into matrices and singular value decomposition (SVD) is utilized to decompose matrices into two factors [30]. However, the reshaping operation leads to distortion of structure information. Hence, more efficient tensor decomposition has attracted interests. Canonical-Polyadic (CP) decomposition [15] is applied to decompose a convolutional layer into four consecutive convolutional layers, significantly speeding up CNNs [12]. Tucker decomposition [31] can decompose the 4D kernel into a 4D compact kernel and two matrices by exploiting the channel-wise redundancy. Based on these three classic decompositions, many other flexible methods emerged including HT [32], TT [33], TR [34], BTD [35], GKPD [36], semi-tensor product (STP) based sem-tensor train (STT) and semi-tensor ring (STR) [37], which dramatically improve the compression performance for DNNs. Table 1 shows the performance of widely-used tensor decomposition methods applied to compress ResNet32 with Cifar10 dataset.

Table 1 Comparison of compression performance of advanced tensor decomposition methods on ResNet32 with Cifar10 dataset

Method	Top-1 Accuracy/%	Compression ratio
Tucker [9]	87.70	5 times
TT [38]	88.3	4.8 times
TR [14]	90.6	5 times
BTD [39]	91.1	5 times
GKPD [36]	91.5	5 times
HT [40]	89.9	1.6 times
STT [37]	91.0	9 times

Here, we outline some key notations. For a fully-connected (FC) layer, we let $\boldsymbol{W} \in \mathbf{R}^{O \times I}$ denote the weight matrix of this layer, where I and O represent the number of input neurons and output neurons, respectively. And for a convolutional (Conv) layer, we let $\boldsymbol{K} \in \mathbf{R}^{S \times C \times H \times W}$ denote the weight of the convolutional kernel, where S, C are the number of filters and input channels, and H, W are the height and width of the kernel. In some cases, we need to reshape a tensor into a higher-order one. We assume that $I_1 \times I_2 \times \cdots \times I_d = I$, $O_1 \times O_2 \times \cdots \times O_d = O$, $C_1 \times C_2 \times \cdots \times C_d = C$, and $S_1 \times S_2 \times \cdots \times S_d = S$. Some necessary mathematical operators are listed in Table 2.

Table 2 Notations used in this paper

Notation	Description
$\mathrm{diag}(\cdot)$	Generation of a diagonal matrix by taking the input vector as the main diagonal
$\otimes$	Kronecker product
$\circ$	Vector outer product
$\times_n$	n-mode product
$\ltimes$	Semi-tensor product

Base on these defined notation, we can make a comparison among various state-of-art tensor decompositions on their ability to compress and accelerate. When aiming at FC layers, the comparison is shown in Table 3. And Table 4 is for Conv layers. Note that in Table 3 $I_m = \max_{\{k\in 1,2,\cdots,d\}} I_k$, $O_m = \max_{\{k\in 1,2,\cdots,d\}} O_k$, $d=2$ for KPD, r is the maximal rank, R is the CP rank of BTD, and t is the ratio between connected dimensionality. Note that in Table 4, $C_m = \max_{\{k\in 1,2,\cdots,d\}} C_k$, $S_m = \max_{\{k\in 1,2,\cdots,d\}} S_k$, $d=2$ for GKPD, $k=\max(k_1,k_2)$ with $k_1\cdot k_2 = K$, r is the maximal rank, R is the CP rank of BTD, M and N are the height and width of feature map, and t is the ratio between connected dimensionality.

Table 3 Comparison among FC layer compressed by TT, TR, HT, BTD, STR, and KPD on computation costs and storage consumption

Method	Computation	Storage
FC	$O(IO)$	$O(IO)$
TT	$O(dI_m\max(I,O)r^2)$	$O(dI_mO_mr^2)$
TR	$O(d(I+O)r^3)$	$O(d(I_m+O_m)r^2)$
HT	$O(d\min(I,O)(r^3+I_mr^2))$	$O(dI_mO_mr+dr^3)$
BTD	$O(dI_m\max(I,O)r^dR)$	$O((dI_mO_mr+r^d)R)$
STR	$O\left(\frac{d(I+O)r^3}{t}\right)$	$O\left(\frac{d(I_m+O_m)r^2}{t^2}\right)$
KPD	$O(IO_m+OI_m)$	$O(I_mO_m)$

Table 4 Comparison among convolutional layer compressed by TT, TR, HT, BTD, STR, GKPD on computation costs and storage consumption.

Method	Computation	Storage
Conv	$O(SCK^2MN)$	$O(SCK^2)$
TT	$O(dr\max(rC_m,K^2)\max(C,S)MN)$	$O(dC_mS_mr^2+K^2r)$
TR	$O(r^3(C+S)+(r^3K^2+r^2(C+S))MN)$	$O((dC_mS_m+K^2)r^2)$
HT	$O(\log_2 dCS(r^3+r^2)+SCK^2MN)$	$O(dC_mS_mr+K^2r+dr^3)$
BTD	$O((K^2r^2+(C+S)r)RMN)$	$O((K^2r^2+(I+O)r)R)$
STR	$O\left(\frac{r^3}{t^3}(C+S)+(r^3K^2+\frac{r^2}{t}(C+S))MN\right)$	$O\left(\left(\frac{dC_mS_m}{t^2}+K^2\right)r^2\right)$
GKPD	$O(r(C_mS+S_mC)k^2MN)$	$O(rC_mS_mk^2)$

2.1.1 SVD

For a given matrix $\boldsymbol{X}\in\mathbf{R}^{M\times N}$, its SVD can be written as

$$\boldsymbol{X}=\boldsymbol{U}\,\mathrm{diag}(\boldsymbol{s})\boldsymbol{V}^{\mathrm{T}}. \tag{1}$$

Let R denote the rank of the matrix, $R\leqslant\min\{M,N\}$. Note that $\boldsymbol{U}\in\mathbf{R}^{M\times N}$ and $\boldsymbol{V}\in\mathbf{R}^{N\times R}$ satisfy $\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}}=\boldsymbol{I}$ and $\boldsymbol{V}\boldsymbol{V}^{\mathrm{T}}=\boldsymbol{I}$, respectively. $\boldsymbol{s}\in\mathbf{R}^R$ is referred to as the singular value vector, where the elements decrease from first to end, i.e., $s_1\geqslant s_2\geqslant\cdots\geqslant s_R$.

Since the format of weights in FC layers is a natural matrix, SVD can be directly utilized. By using SVD, the FC layer can be approximated by two consecutive layers. The weight of the first and second layer can be represented by $\boldsymbol{B}=\mathrm{diag}(\sqrt{\boldsymbol{s}})\boldsymbol{V}^{\mathrm{T}}$ and $\boldsymbol{A}=\boldsymbol{U}\mathrm{diag}(\sqrt{\boldsymbol{s}})$, respectively. For Conv layers, the 4D kernel should be reshaped into a two-dimensional (2D) matrix first. By exploiting different types of redundancy, there are two decomposition schemes. One reshapes $\mathcal{W}$ into a S-by-$C\cdot H\cdot W$ matrix, namely channel-wise decomposition [30]. The other called spatial-wise decomposition [13] gets a $S\cdot H$-by-$C\cdot W$ matrix. Then, compute SVD of the reshaped matrix. Similar to the process of compressing FC layers, two Conv layers represented by tensors reshaped from factors $\boldsymbol{B}$ and $\boldsymbol{A}$ can be used to replace the original layer.

However, both methods only can exploit one type of redundancy. Moreover, there is also redundancy between input channels. Exploiting all kinds of redundancy at the same time can help us achieve a much higher compression ratio, which can be achieved by tensor decomposition.

2.1.2 CP decomposition

While SVD factorizes a matrix into a sum of rank-one matrices, CP decomposition factorizes a tensor into a sum of rank-one tensors. For an Nth order tensor, $\boldsymbol{X}\in\mathbf{R}^{I_1\times I_2\times\cdots\times I_N}$, the CP decomposition can be formulated as:

$$\boldsymbol{X}=\left[\boldsymbol{\lambda};\boldsymbol{A}^{(1)},\boldsymbol{A}^{(2)},\cdots,\boldsymbol{A}^{N}\right]=\sum_{r=1}^{R}\lambda_r\boldsymbol{a}_r^{(1)}\circ\boldsymbol{a}_r^{(2)}\circ\cdots\circ\boldsymbol{a}_r^{(N)}. \tag{2}$$

Each $\boldsymbol{a}_r^{(n)}$ represents the rth column of $\boldsymbol{A}^{(n)}$ and $\boldsymbol{\lambda}\in\mathbf{R}^R$ represents the significance of R components. The rank of the tensor $\boldsymbol{X}$, denoted by R, is defined as the smallest number of rank-one tensors [27, 41].

When using CP to compress FC layers, the weight matrix $\boldsymbol{W}$ should be firstly tensorized into a $2d$th order tensor $\boldsymbol{W}'\in\mathbf{R}^{O_1\times O_2\cdots\times O_d\times I_1\times I_2\times\cdots\times I_d}$. Meanwhile, the input vector $\boldsymbol{x}\in\mathbf{R}^I$ should be presented as a dth order tensor $\boldsymbol{X}\in\mathbf{R}^{I_1\times I_2\times\cdots\times I_d}$. For convolutional kernels, by directly performing CP on the 4D kernel tensor, the layer will be approximated by four consecutive convolutional layers whose weights are represented by four factor matrices, respectively.

2.1.3 Tucker decomposition

The Tucker decomposition can be considered as a higher-order generalization of principal component analysis (PCA). It represents an Nth order tensor with a Nth order core tensor multiplied by a basis matrix along each mode. Thus, for $\boldsymbol{X} \in \mathbf{R}^{I_1 \times I_2 \times \cdots \times I_N}$, we have

$$\boldsymbol{X} = \boldsymbol{G} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \times_3 \cdots \times_N \boldsymbol{A}^{(N)} \tag{3}$$

where $\boldsymbol{G} \in \mathbf{R}^{R_1 \times R_2 \times \cdots \times R_N}$ is called core tensor. Elementwise, "$\times_n$" can be formulated as

$$(\boldsymbol{G} \times_1 \boldsymbol{A}^{(1)})_{i_1, r_2, \cdots, r_N} = \sum_{r_1=1}^{R_1} \boldsymbol{G}_{r_1, r_2, \cdots, r_N} \boldsymbol{A}^{(1)}_{i_1, r_1}. \tag{4}$$

Columns of the factor matrix $\boldsymbol{A}^{(n)} \in \mathbf{R}^{I_n \times R_n}$ can be considered as the principal components of the nth mode. The core tensor $\boldsymbol{G}$ can be viewed as a compressed version of $\boldsymbol{X}$ or the coefficient in the low dimensional subspace. In this case, we can say that $\boldsymbol{X}$ is a rank-$(R_1, R_2, \cdots, R_N)$ tensor [27,41].

In the case of compressing FC layers, similar to CP, the same tensorization for weights and input is needed, followed by directly performing Tucker decomposition on the $2d$th order tensor. For Conv layers, since the spatial size of the kernel is too small, we can just use Tucker2 [42] to take advantage of redundancy between filters and between input channels, generating 1×1 convolution, $H \times W$ convolution, and 1×1 convolution.

2.1.4 BTD

BTD was introduced in [35] as a more powerful tensor decomposition, which combines the CP decomposition and Tucker decomposition. Consequently, BTD is more robust than the original CP and Tucker decomposition. While CP approximates a tensor with a sum of rank-one tensors, BTD is a sum of tensors in low rank Tucker format. Or, by concatenating factor matrices in each mode and arranging all the core tensors of each subtensor into a block diagonal core tensor, BTD can be considered as an instance of Tucker. Hence, consider a Nth order tensor, $\boldsymbol{X} \in \mathbf{R}^{I_1 \times I_2 \times \cdots \times I_d}$, its BTD can be written as

$$\boldsymbol{X} = \sum_{n=1}^{N} \boldsymbol{G}_n \times_1 \boldsymbol{A}_n^{(1)} \times_2 \boldsymbol{A}_n^{(2)} \times_3 \cdots \times_d \boldsymbol{A}_n^{(d)}. \tag{5}$$

In (5), N denotes the CP rank, i.e., the number of block terms, and $\boldsymbol{G}_n \in \mathbf{R}^{R_1 \times R_2 \times \cdots \times R_d}$ is the core tensor of the nth block term with multilinear ranks that equals $(R_1, R_2, \cdots, R_d)$.

When BTD is applied to compress an FC layer, the yielded compact layer is called block term layer (BTL) [39]. In the BTL, the input tensor $\boldsymbol{X} \in \mathbf{R}^{I_1 \times I_2 \times \cdots \times I_d}$ is tensorized from the original input vector $\boldsymbol{x} \in \mathbf{R}^I$ and the original weight matrix $\boldsymbol{W}$ is reshaped as $\boldsymbol{W}' \in \mathbf{R}^{O_1 \times I_1 \times O_2 \times I_2 \times \cdots \times O_d \times I_d}$. Then, we can factorize $\boldsymbol{W}'$ by BTD with factor tensors $\{\boldsymbol{A}_n^{(d)} \in \mathbf{R}^{O_d \times I_d \times R_d}\}_{n=1}^{d}$. By conducting a tensor contraction operator between BTD($\boldsymbol{W}'$) and $\boldsymbol{X}$, the output tensor $\boldsymbol{Y} \in \mathbf{R}^{O_1 \times O_2 \cdots \times O_d}$ comes out, which can be vectorized as the final output vector. For Conv layers, it is claimed in [39] that by reshaping the 4D kernel into a matrix, $\boldsymbol{W} \in \mathbf{R}^{S \times C \cdot H \cdot W}$, the layer can be transformed into BTL. Specifically speaking, the matrix should be further reshaped as $1 \times H \times 1 \times W \times S_1 \times C_1 \times S_2 \times C_2 \times \cdots \times S_d \times C_d$.

2.1.5 HT decomposition

HT decomposition is a hierarchical variant of the Tucker decomposition, which iteratively represents a high-order tensor with two lower-order subtensors and a transfer matrix via taking advantage of the Tucker decomposition [32,43]. For a tensor $\boldsymbol{X} \in \mathbf{R}^{I_1 \times I_2 \times \cdots \times I_N}$, we can simply divide the index set $\{1, 2, \cdots, N\}$ into two subsets, i.e., $T = \{t_1, t_2, \cdots, t_k\}$, $S = \{s_1, s_2, \cdots, s_{N-k}\}$. Let $\boldsymbol{U}_{12\cdots N} \in \mathbf{R}^{I_{t_1} I_{t_2} \cdots I_{t_k} I_{s_1} I_{s_2} \cdots I_{s_{N-k}} \times 1}$ denote the matrix reshaped from $\boldsymbol{X}$, and truncated matrices $\boldsymbol{U}_t \in \mathbf{R}^{I_{t_1} I_{t_2} \cdots I_{t_k} \times R_t}$, $\boldsymbol{U}_s \in \mathbf{R}^{I_{s_1} I_{s_2} \cdots I_{s_{N-k}} \times R_s}$ represent the corresponding column basis matrix of two subspaces. Then, we can have

$$\boldsymbol{U}_{12\cdots N} = (\boldsymbol{U}_t \otimes \boldsymbol{U}_s) \boldsymbol{B}_{12\cdots N} \tag{6}$$

where $\boldsymbol{B}_{12\cdots N} \in \mathbf{R}^{R_t R_s \times 1}$ is termed as transfer matrix and "$\otimes$" denotes the Kronecker product between two matrices. Subsequently, divide the set T into two subsets $L = \{l_1, l_2, \cdots, l_q\}$ and $V = \{v_1, v_2, \cdots, v_{k-q}\}$. We can represent $\boldsymbol{U}_t$ with $\boldsymbol{U}_l \in \mathbf{R}^{I_{l_1} I_{l_2} \cdots I_{l_q} \times R_l}$, $\boldsymbol{U}_v \in \mathbf{R}^{I_{v_1} I_{v_2} \cdots I_{v_{k-q}} \times R_v}$, and $\boldsymbol{B}_t \in \mathbf{R}^{R_l R_v \times R_t}$ as

$$\boldsymbol{U}_t = (\boldsymbol{U}_l \otimes \boldsymbol{U}_v) \boldsymbol{B}_t. \tag{7}$$

The similar factorization procedure applies simultaneously to $\boldsymbol{U}_s$. By repeating this procedure until the index set cannot be divided, we can eventually obtain the tree-like HT format of the target tensor. An illustration of a simple version of HT can be seen in Fig. 2.

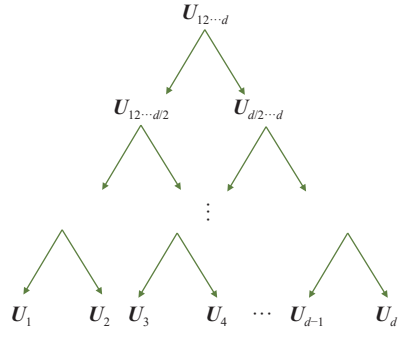

Fig. 2 HT decomposition

Since the Kronecker product in (7) is computationally expensive, there are other concise forms of HT, such as the contracted form introduced in [40]. This form merges index subsets to the universal set from bottom to top. In this form, an external input can be contracted with each transfer matrix and truncated matrix one by one. This way can avoid the memory and computation-consuming weight reconstruction procedure and intermediate outputs will not be too large to out of memory.

For the realization of compressing FC layers by HT, the weight matrix should be transformed into $\boldsymbol{W}' \in \mathbf{R}^{(I_1 \cdot O_1)\times(I_2 \cdot O_2)\times\cdots\times(I_d \cdot O_d)}$, and the input data is tensorized into $\boldsymbol{X} \in \mathbf{R}^{I_1\times I_2\times\cdots\times I_d}$. For reducing computation complexity, the chain computation shown in Fig. 3 is applied. However, as there is no law associating convolution and contraction, the kernel of Conv layers must be recovered from the HT format. By the way, in order to keep balance, the 4D kernel should be tensorized into $\boldsymbol{W} \in \mathbf{R}^{(H\cdot W)\times(C_1\cdot S_1)\times(C_2\cdot S_2)\times\cdots\times(C_d\cdot S_d)}$.

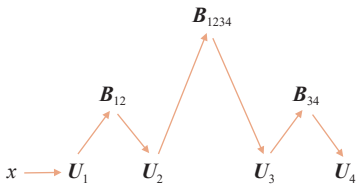

Fig. 3 The chain computation for a fourth-order case

2.1.6 TT decomposition

TT is a special case of HT, which is a degenerate HT format [33,44]. TT factorizes a high-order tensor into a collection of third-or second-order tensors. These core tensors are connected by the contraction operator. Assume that we have a Nth order tensor, $\boldsymbol{X} \in \mathbf{R}^{I_1\times I_2\times\cdots\times I_N}$, elementwise, we can factorize it into TT format as

$$\mathcal{X}_{i_1,i_2,\cdots,i_N} = \sum_{r_1,r_2,\cdots,r_N} \mathcal{G}^1_{i_1,r_1}\mathcal{G}^2_{r_1,i_2,r_2}\cdots\mathcal{G}^N_{r_{N-1},i_N} \tag{8}$$

where the collection of $\{\boldsymbol{G}^n \in \mathbf{R}^{R_{n-1}\times I_n\times R_n}\}_{n=1}^N$ with $R_0 = 1$ and $R_N = 1$ is called TT-cores [33]. The collection of ranks $\{R_n\}_{n=0}^N$ is called TT-ranks. Fig. 4 gives an illustration of a fourth order tensor represented in TT format. $\boldsymbol{X}$ represents a fourth-order input. These arrows represent the order of contraction.

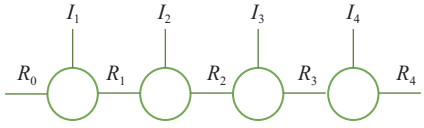

Fig. 4 A fourth order tensor in TT format

The TT was first applied to compress FC layers in [45], where the weight matrix is reshaped into a high order tensor, $\boldsymbol{W}' \in \mathbf{R}^{(I_1\cdot O_1)\times(I_2\cdot O_2)\times\cdots\times(I_d\cdot O_d)}$. After representing $\boldsymbol{W}'$ in TT format, the resulted TT-cores $\{\boldsymbol{G}^n \in \mathbf{R}^{R_{n-1}\times I_n\times O_n\times R_n}\}_{n=1}^N$ can directly be contracted with the tensorized input. It was suggested in [40] that TT is more efficient for compressing Conv layers than HT, while HT is more suitable for compressing FC layers whose weight matrix is more prone to be reshaped into a balanced tensor.

Employing TT on Conv layers is introduced in [38], where the 4D kernel tensor should be reshaped to size of $(H\cdot W)\times(C_1\cdot S_1)\times(C_2\cdot S_2)\times\cdots\times(C_d\cdot S_d)$ and the input feature maps are reshaped to $\boldsymbol{X} \in \mathbf{R}^{H\times W\times C_1\times\cdots\times C_d}$. In the feedforward phase, the tensorized input $\boldsymbol{X}$ will be contracted with each TT-core one by one. Although TT can significantly save storage costs, the computational complexity may be higher than the original Conv layer. Hence, high-oeder decomposed convolution (HODEC) was proposed in [46] to enable simultaneous reductions in computational and storage costs, which further decomposes each TT-cores into two third-order tensors.

2.1.7 TR decomposition

Due to the disunity of edge TT-cores, there is still an open issue that how to arrange dimensions of tensors to find the optimal TT format. To conquer this problem, TR decomposition was proposed to perform a circular multilinear product over cores [34, 47−49]. Consider a given tensor, $\boldsymbol{X} \in \mathbf{R}^{I_1\times I_2\times\cdots\times I_N}$, elementwise, we can formulate its TR representation as

$$\mathcal{X}_{i_1,i_2,\cdots,i_N} = \sum_{r_1,r_2,\cdots,r_N} \mathcal{G}^1_{r_1,i_1,r_2}\mathcal{G}^2_{r_2,i_2,r_3}\cdots\mathcal{G}^N_{r_N,i_N,r_1} = \operatorname{tr}\left(\sum_{r_2,\cdots,r_N} \mathcal{G}^1_{:,i_1,r_2}\mathcal{G}^2_{r_2,i_2,r_3}\cdots\mathcal{G}^N_{r_N,i_N,:}\right) \tag{9}$$

where all cores $\{\boldsymbol{G}^n \in \mathbf{R}^{R_n\times I_n\times R_{n+1}}\}_{n=1}^N$ with $R_{N+1} = R_1$ are called TR-cores. Its tensor diagram for a fourth-order tensor is illustrated in Fig. 5. This form is equivalent to the sum of R_1 TT format. Thanks to the circular multilinear product gained by employing the trace operation, TR treats all the cores equivalently and becomes much more powerful and general than TT.

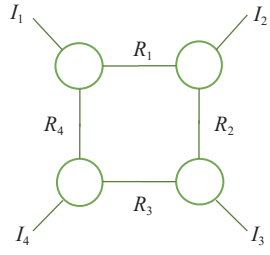

Fig. 5 A fourth-order tensor in TR format

Moreover, due to the circular strategy, TR amends the variousness of gradients in TT. Hence, TR is also suit-

able for compressing FC layers. In [14], TR was first applied to compress DNNs. Specifically speaking, the weight matrix of FC layers should be reshaped into a $2d$th order tensor of size $I_1 \times I_2 \times \cdots \times I_d \times O_1 \times O_2 \times \cdots \times O_d$, followed by representing the tensor into TR format. For the feedforward process, firstly, merge the first d cores and the last d cores to obtain $\boldsymbol{F}_1 \in \mathbf{R}^{R_1 \times I_1 \times \cdots \times I_d \times R_{d+1}}$ and $\boldsymbol{F}_2 \in \mathbf{R}^{R_{d+1} \times O_1 \times \cdots \times O_d \times R_1}$, respectively. Then, we can calculate contraction between input $\boldsymbol{X} \in \mathbf{R}^{I_1 \times I_2 \times \cdots \times I_d}$ and $\boldsymbol{F}_1$, yielding a matrix that can be contracted with $\boldsymbol{F}_2$. The final output tensor will be of size $O_1 \times O_2 \times \cdots \times O_d$. For Conv layers, if keeping the kernel tensor in 4th order and maintaining the spatial information, its TR-format can be formulated as

$$\mathcal{K}_{s,c,h,w} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \sum_{r_3=1}^{R_3} \mathcal{U}_{r_1,s,r_2} \mathcal{V}_{r_2,c,r_3} \mathcal{Q}_{r_3,h,w,r_1}. \tag{10}$$

Hence, the original layer can be represented by three consecutive layers whose weight tensors are $\boldsymbol{V}$, $\boldsymbol{Q}$, and $\boldsymbol{U}$ respectively. If a higher compression ratio is needed, we can further view $\boldsymbol{U}$ and $\boldsymbol{V}$ as tensors merged from d core tensors respectively, with an extra computation burden of merging.

2.1.8 Generalized KPD

KPD can factorize a matrix into two smaller factor matrices interconnected by Kronecker product, which has shown to be very effective when applied to compress RNNs [50]. To further compress Conv layers, it was generated to generalized KPD (GKPD) [36], which represents a tensor by the sum of multidimensional Kronecker product between two factor tensors. Formally, the multidimensional Kronecker product between $\boldsymbol{U} \in \mathbf{R}^{J_1 \times J_2 \times \cdots \times J_N}$ and $\boldsymbol{V} \in \mathbf{R}^{K_1 \times K_2 \times \cdots \times K_N}$ is formulated as

$$(\boldsymbol{U} \otimes \boldsymbol{V})_{i_1,i_2,\cdots,i_N} = \boldsymbol{U}_{j_1,j_2,\cdots,j_N} \boldsymbol{V}_{k_1,k_2,\cdots,k_N} \tag{11}$$

where $j_n = \lfloor i_n / K_n \rfloor$ and $k_n = i_n \bmod K_n$. Based on this, for a given Nth order tensor $\boldsymbol{\chi} \in \mathbf{R}^{J_1 K_1 \times J_2 K_2 \times \cdots \times J_N K_N}$, GKPD can be denoted as

$$\boldsymbol{\chi} = \sum_{r=1}^{R} \boldsymbol{U}_r \otimes \boldsymbol{V}_r \tag{12}$$

where R is referred to as Kronecker rank. For finding the best approximation in GKPD with a given R, we can transform this optimization problem into finding a best rank-R approximation for a matrix, which can be solved by SVD conveniently, via carefully rearranging $\boldsymbol{\chi}$ into a matrix and rearranging $\boldsymbol{U}$ and $\boldsymbol{V}$ into vectors.

For the realization of using GKPD to decompress Conv layers, the 4D kernel is represented as

$$\mathcal{W}_{S,C,H,W} = \sum_{r=1}^{R} (\boldsymbol{U}_r)_{S_1,C_1,H_1,W_1} \otimes (\boldsymbol{V}_r)_{S_2,C_2,H_2,W_2} \tag{13}$$

where $S_1 S_2 = S$, $C_1 C_2 = C$, $H_1 H_2 = H$, and $W_1 W_2 = W$. The 2D convolution between each $\boldsymbol{U}_r \otimes \boldsymbol{V}_r$ and input can be transformed into a three-dimensional (3D) convolution whose depth equals C_2, followed by multiple 2D convolutions. Furthermore, the group of R Kronecker products can be viewed as R parallel channels that calculate the above two steps separately. And it was analysed that large S_1 and C_2 can help to obtain more reduction in FLOPs.

2.1.9 STP-based tensor decomposition

STP [51] is a generation of the conventional matrix product, which extends the calculation of two dimensionally matching matrices to that of two dimensionally arbitrary matrices. Since tensor contraction is based on the conventional matrix product, we can further substitute STP into tensor contraction, which will lead to more general and flexible tensor decomposition methods. In [37], STP-based tensor decomposition was designed to enhance the flexibility of Tucker decomposition, TT and TR by replacing the conventional matrix product in tensor contraction by STP, which demonstrates much higher efficiency than original methods. Consider a special case in which the number of columns in $\boldsymbol{X} \in \mathbf{R}^{M \times NP}$ is proportional to that of rows in $\boldsymbol{W} \in \mathbf{R}^{P \times Q}$, the STP can be denoted as

$$\boldsymbol{Y} = \boldsymbol{X} \ltimes \boldsymbol{W}, \tag{14}$$

or, elementwise, as

$$\boldsymbol{Y}_{m,g(n,q)} = \sum_{p=1}^{P} \boldsymbol{X}_{m,g(n,p)} \boldsymbol{W}_{p,q}. \tag{15}$$

Note that $\boldsymbol{Y} \in \mathbf{R}^{M \times NQ}$, "$\ltimes$" denotes the STP, $g(n,q) = (q-1)N + n$, and $g(n,p) = (p-1)N + n$ are reindexing functions.

Hence, take STP-based Tucker decomposition as an example, namely semi-tensor Tucker (STTu) decomposition, which can be formulated as

$$\mathcal{X} = \mathcal{G} \ltimes_1 \boldsymbol{A}^{(1)} \ltimes_2 \boldsymbol{A}^{(2)} \ltimes_3 \cdots \ltimes_N \boldsymbol{A}^{(N)} \tag{16}$$

where $\mathcal{G} \in \mathbf{R}^{R_1 \times R_2 \times \cdots \times R_N}$, $\boldsymbol{A}^{(n)} \in \mathbf{R}^{\frac{I_n}{t} \times \frac{R_n}{t}}$. Compared with normal Tucker, the number of parameters is reduced from $\left(\prod_{n=0}^{N} R_n + \sum_{n=1}^{N} I_n R_n \right)$ to $\left(\prod_{n=0}^{N} R_n + \sum_{n=1}^{N} \frac{I_n R_n}{t^2} \right)$.

2.2 Low rank optimization method

We have already introduced various tensor decomposition methods, but how to apply these methods to DNNs without significant accuracy degradation is an optimization problem, which remains to be discussed. Since the

lower the tensor rank is, the higher compression ratio we will get, we hope that each layer of DNNs can be decomposed by very low rank tensor decomposition. However, as the rank gets lower, the approximation error increases, leading to a dramatic loss of accuracy. Hence, there is a tradeoff between accuracy and compression ratio, which is a widely studied problem. There are mainly three kinds of low rank optimization methods to achieve a good tradeoff: pre-train method, pre-set method and compression-aware method (representative works can be seen in Table 5). For each method, we give the key points about the implementation in detail.

Table 5 Three types of low rank optimization method for model compression

Method	Description	Representative works
Pre-train	Pretrain the target model, apply tensor decomposition to trained weight tensors, and then fine-tune to recover accuracy	[9, 12, 30, 52]
Pre-set	Construct tensorized netwoks, set proper initialization, and then train the whole network	[14, 38,39]
Compression-aware	Train the original network with normal optimizers but enforce weight tensors to enjoy low rank structure	[53−55]

2.2.1 Pre-train method

The pre-train method is the earliest used method in the literature of applying tensor compression to model compression, which directly decomposes an already trained network into a compact format, followed by fine-tuning to recover the accuracy. There are two critical issues for implementation: tensor rank selection and instability.

Tensor rank selection means how to select the proper tensor rank of each layer in a network. Since the extent of redundancy varies from one layer to another, the rank of each layer is not supposed to be equal. Hence, unlike time-consuming trial-and-error, an efficient rank selection method should allocate the limited computation or storage resources to each layer reasonably via carefully deciding the rank of each layer, while ensuring the lowest accuracy degradation.

A simple but effective way is to set the rank of each layer to be proportional to the number of corresponding input or output channels, which usually performs better than roughly setting all ranks equal. A probabilistic matrix factorization tool called variational Bayesian matrix factorization (VBMF) [56] was used in [9] to estimate tensor ranks of a tensor in Tucker format. In order to get the mode-n rank, the corresponding mode-n matricization of the target tensor was viewed as an observation with noise. Then, VBMF was employed on the observation to filter out the noise and then obtain a low rank matrix. In [30], the rank selection problem was formulated as a combinatorial optimization problem [57] with computation or memory resource constrained. The objective function is denoted as the product of PCA energy (the sum of singular values) of each layer, as the authors empirically observe that the PCA energy is roughly related to the classification accuracy. Similarly, the algorithm in [52] also employed the idea that the approximation error is linked to the accuracy loss. But more efficiently and reasonably, it selects the maximum approximation error of all the layers as a proxy for model accuracy. By minimizing this proxy, it is guaranteed that no layer decomposed will significantly reduce the accuracy. Together with the resource constraint, the final problem is a minimax optimization which can be solved by binary search.

Since the approximation error does not necessarily reflect the loss of accuracy, the above methods can only obtain a suboptimal rank configuration scheme. To address this challenge, reinforcement learning is employed to automatically select ranks [58,59]. In the established state-action-reward system, the reward favors a reduction in resource cost and penalizes loss of accuracy. The state (a possible global rank configuration of all the layers) that renders the maximum reward can be chosen as the next state.

Instability means that if a model is approximated by an unstable decomposed format such us CP format and TR format, it will lead to difficulty in fine-tuning, i.e., converge slowly and converge to a false local minima. In [60−62], it was noted that there is a degeneracy problem that causes instability in CP decomposition. Specifically speaking, when CP represents a relatively high-rank tensor in a low-rank format, there are at least two rank-one components whose Frobenius norm goes to infinity and cancels each other out. Due to the instability, [12,63] fails to decompose the whole network by CP decomposition, as it is difficult to find a suitable fine-tuning learning rate. To deal with this challenge, [64] proposed to use the tensor power method [65] to calculate CP decomposition and employ iterative fine-tuning, i.e., decomposed one layer at a time and then fine-tune the entire network iteratively. The authors of [66] devise a procedure to minimize the sensitivity (a measure for the degeneracy degree) of the tensor reconstructed from CP format so that the decomposed network with low sensitivity can be fine-tuned faster and obtains a better accuracy. A more

direct method proposed in [67] hold that each column of the factor matrix should be normalized after each update, as normalization can improve numerical stability in the outer vector product [68]. A similar instability problem also happened to TR [69]. Hence, [70] proposed a sensitivity correction procedure to address the problem via minimizing the sensitivity with an approximation error bounded constraint.

2.2.2 Pre-set method

The pre-set method has the interpretation that a tensorized neural network that is preset to a low tensor rank format will be trained from scratch. As the method requires no pre-training, it can save a great deal of time to get a compressed model. However, the method is sensitive to initialization and difficult to achieve high accuracy due to the limited model capacity. Moreover, similar to the pre-train method, there are also problems in configuring ranks. In a nutshell, proper initialization and tensor rank selection are the main issues with this method.

Initialization plays an important role in providing a warm start for training DNNs [71] as well as for the training of low rank structure networks [14], and can have an impact on the final accuracy to a large extent. An empirically determined suitable initialization distribution for weights in a layer is $\mathrm{N}\left(0, \mathrm{std} = \sqrt{\frac{2}{N}}\right)$, where N denotes the total number of parameters in this layer. For a pre-set model, we should make sure that weights in each layer approximated by factor tensors also obey this distribution. For example, when a layer is compressed by TR and the distribution of each core tensor is $\mathrm{N}(0, \sigma^2)$, then after merging these d core tensors, elements of the merged tensor will have mean 0 and variance $R^d\sigma^{2d}$. Hence, we need to set σ^2 as $\left(\frac{2}{N}\right)^{\frac{1}{d}} R^{-1}$ to obtain a good initialization. Similarly, for TT, the variance of TT-cores should be $\left(\frac{2}{N}\right)^{\frac{1}{d}} R^{\frac{1}{d}-1}$. A more systematic analysis of initialization for any tensor decomposition method was introduced in [72]. It is suggested that by extracting the Backbone structure (i.e., a structure only contains contracted dimensions, since only the contraction operator will change the variance of weights) from the original tensorized structure, an adjacency matrix can be obtained from node edges of the Backbone structure, which can be utilized to adjust the variance of factor tensors.

Tensor rank selection is seldom studied in the works of training a tensorized neural network and usually set the ranks to equal in experiments, as it is difficult to verify the redundancy in each layer without a pre-training network. At present, there are only a few methods to solve this problem for specific tensor decompositions. Inspired by neural architecture search (NAS) [73,74] proposes a progressive searching TR network (PSTRN), which has the ability to find an appropriate rank configuration for TR efficiently. In this algorithm, an evolutionary phase and a progressive phase are alternatively performed. While the evolutionary phase is responsible for deriving good rank choices within the search space via multi-objective genetic algorithm i.e., non-dominated sorting genetic algorithm-II (NSGA-II) [75], the progressive phase is responsible for narrowing the search space in the vicinity of the optimized rank coming from the previous evolutionary phase. For rank selection with TT decomposition, [76] proposes a low-rank Bayesian tensorized neural network. Bayesian methods are always used to infer tensor ranks in CP format or Tucker format through low-rank priors in tensor completion tasks [77−79]. This paper generates this approach to TT format and nonlinear neural networks.

A more easily implemented method, modified Beam-search, was proposed in [80] to find the optimal rank setting, costing much lower search time than the full search. To verify optimality, it adopts the validation accuracy on a mini-batch validation dataset as its metric. This method is applicable to all kinds of tensor decompositions.

2.2.3 Compression-aware method

Compression-aware method is the method that through standard training and iterative optimization, the weights of kernels and FC layers can gradually have desired low tensor rank structures. That is, consider the future compression into the standard training phase. Upon the end of this one-shot training, the suitable tensor ranks are automatically learned, without efforts to design efficient rank selection schemes. Moreover, since the training process is still on the original network structure instead of a deeper factorized network, it's easy to converge towards high accuracy without being prone to gradient vanishing or explosion. There are mainly two kinds of ways to realize this idea, namely using low rank regularization and solving constrained optimization.

Low rank regularization is similar to the sparse regularization which is always used in DNNs to avoid overfitting. The main idea of low rank regularization is to add low rank regularizer on weights in each layer to the basic loss function. Hence, with the constraint of such regularizer, weight tensors will gradually have a desired low rank structure during training. Then, after low rank approximation, there is no need to retrain for a long time and no risk of unstable recovery.

For the low rank regularizer, an index to measure the

low rank degree is essential. Since explicitly minimizing the rank of a matrix is NP-hard, nuclear norm [81] was widely used as a continuous convex relaxation form of rank. In [82], the sum of nuclear norms of weight matrices in each layer was added to cross-entropy loss, yielding a new optimization problem which can be solved by proximal stochastic gradient descent. Similarly, [83] also used nuclear norm and the same optimization problem was solved by stochastic sub-gradient descent [84]. In addition, this paper embeds the low rank approximation into the training phase to boost the low rank structure.

However, for the above, SVD will be performed on every training step, which is inefficient, especially for larger models. Hence, [85] proposed SVD training which performs training directly on the decomposed factors. By employing sparsity regularization on singular values, it can achieve the goal of boosting low rank. In order to maintain the valid SVD form, orthogonality regularization on the left and right singular matrices is necessary. Moreover, Orthogonality also can efficiently prevent the gradient to explode or vanish, therefore achieving higher accuracy.

Solving constrained optimization is a method that through solving an optimization problem with explicit or implicit constraints on tensor ranks of weights, we can get an optimal network not only with low loss but also with low rank structures. Classically, [53] formed the low rank constrained problem as minimizing the sum of the loss and a memory/computation cost function but constraining each rank not to exceed a maximum rank. It can be solved by a learning-compression algorithm [86]. More conveniently, [55] directly used budget (e.g., memory/computation cost) as constraints, with low rank regularizer added on the loss function. However, since it represents tensor ranks by the sum of nuclear norms of unfolding matrices in each mode, it cannot be generalized to certain decomposition methods such as CP and BTD. And when dealing with high-order tensors, there will be too many auxiliary variables used in the augmented Lagrangian algorithm, which will affect convergence. Without using nuclear norm, [54] just set the upper bound of ranks, therefore it is applicable to various tensor decompositions.

The above methods have an unsatisfactory tradeoff between accuracy and compression. To address this drawback, the Frank Wolfe algorithm was utilized in [87] to optimize network weights with the low-rank constraint. This improvement benefits from the highly structured update directions of Frank Wolfe.

For compression-aware methods, using different sparsity measures as low rank regularizers will greatly impact compression performance. For an instance, it was noted in [85] that the ℓ^1 measure (e.g., nuclear norm) is more suitable for an extremely high compression ratio while Hoyer measure performs better when aiming for a relatively low compression ratio. Hence, it's essential to dig out an efficient sparsity measure that is attractive for any compression ratio. This is exactly the point we want to make below.

2.3 Sparsity measure

Recently, researches on compression-aware method emerge in large numbers and plenty of experiments show that with the premise of using the same tensor decomposition method, compression-aware method can outperform the other two methods [54,55, 85]. Hence, we should pay more attention to it. One thing that has not been fully studied is the sparsity measure used. As the most classical convex relaxation form of rank, nuclear norm (ℓ_1 measure) is widely used. However, there is no evidence that the nuclear norm is a perfect choice. Consequently, a comparison between common sparsity measures should be made. Finding a more efficient measure may greatly improve the compression capability of existing compression-aware algorithms.

2.3.1 Common sparsity measure

For sparse representation problems, the ℓ_0 norm defined as the number of non-zeros is the traditional measure of sparseness. However, since the ℓ_0 norm is sensitive to noise and its derivative contains no information, the ℓ_p norm with $0 < p \leqslant 1$ is introduced to less consider the small elements [88]. For a vector $\boldsymbol{x} \in \mathbf{R}^N$, its ℓ_p norm can be formulated as

$$\ell_p(\boldsymbol{x}) = \left(\sum_{i=1}^{N} |x_i|^p \right)^{\frac{1}{p}}. \tag{17}$$

The ℓ_1 norm, ℓ_p norm with $p = 1$, is the most widely used sparsity measure. Formally, consider a vector $\boldsymbol{x} \in \mathbf{R}^N$, its ℓ_1 norm can be denoted as

$$\ell_1(\boldsymbol{x}) = \sum_{i=1}^{N} |x_i|. \tag{18}$$

The ℓ_1 norm was introduced in [89] as a more practical substitute for the ℓ_0 norm. In addition, in order to better measure sparsity in noisy data, more flexible forms based on ℓ_1 norm were proposed in [90,91], namely sorted ℓ_1 norm and two-level ℓ_1 norm. The sorted ℓ_1 norm is formulated as

$$\ell_1^{\text{sort}}(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i |x_i| \tag{19}$$

where $\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_N \geqslant 0$. In this way, the higher the

magnitude of the element, the larger the penalty on it. More concisely, the two-level ℓ_1 norm only considers two levels of penalty, which can be formulated as

$$\ell_1^{\text{2level}}(\boldsymbol{x}) = \rho \sum_{i \in I_1} |x_i| + \sum_{j \in I_2} |x_j| \tag{20}$$

where $|x_i| \geqslant |x_j|, \forall i \in I_1, \forall j \in I_2$. The index of $\boldsymbol{x}$ is indivied into the two sets I_1 and I_2 and they contains the index of larger elements, while the rest is in I_2.

The Gini Index was initially proposed as a measure of the inequity of wealth [92,93]. Afterward, the utility of Gini Index as a measure of sparsity has been demonstrated in [94,95]. Given a sorted vector $\boldsymbol{x} \in \mathbf{R}^N$ whose elements increases by degrees, i.e., $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_N$, its Gini Index is given by

$$G(\boldsymbol{x}) = 1 - 2 \sum_{i=1}^{N} \frac{x_i}{\|\boldsymbol{x}\|_1} \left(\frac{N - i + \frac{1}{2}}{N} \right). \tag{21}$$

Note that if all elements are equal, i.e., no sparsity, the Gini Index reaches its minimal 0. For the most sparse case, i.e., only x_N is non-zero, the Gini Index goes to a maximum of $1 - \frac{1}{N}$. Graphically, the Gini Index can be represented as twice the area between the Lorenz curve [93] and the 45° line. Each point on the Lorenz curve ($x = a$, $y = b$) has the interpretation that top $100 \times a$ percent of the sorted elements expresses $100 \times b$ percent of the total power. The degree line represents the least sparse case with Gini Index equal to 0. Fig. 6 illustrates the Lorenz curve for a vector. The dot line (45° line) represents the case in which all elements are equal, and the full line is the Lorenz curve of the vector. Twice the area between them is equal to the Gini Index of such a vector.

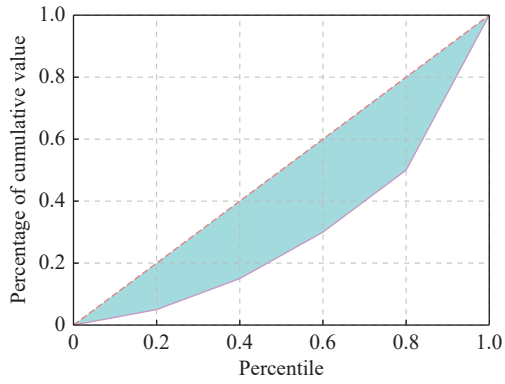

Fig. 6 A graphical illustration of Gini Index for a vector [1,2,3,4,10]

The Hoyer measure was devised in [96] as a new sparsity measure, which is a normalized version of ℓ_2/ℓ_1. For a given vector $\boldsymbol{x} \in \mathbf{R}^N$, its Hoyer measure can be formulated as

$$H(\boldsymbol{x}) = \frac{\sqrt{N} - \frac{\|\boldsymbol{x}\|_1}{\|\boldsymbol{x}\|_2}}{\sqrt{N} - 1}. \tag{22}$$

This function goes to unity if and only if $\boldsymbol{x}$ contains only a single non-zero component, and takes a value of zero if and only if all components are equal, changing smoothly between the two extremes.

The above-mentioned sparsity measure can be applied to the singular value vector as a low rank measure of the corresponding matrix. There are other non-strict measures for the rank of a matrix. Here, we concentrate on effective rank [97]. Let us consider a matrix $\boldsymbol{X}$ of size $M \times N$ whose singular value vector is denoted by $\boldsymbol{\sigma} \in \mathbf{R}^K$ with $K = \min\{M, N\}$, then its effective rank can be given by

$$E(\boldsymbol{X}) = \exp\left(-\sum_{i=1}^{K} \bar{\sigma}_i \ln \bar{\sigma}_i \right) \tag{23}$$

where σ_i is the ith element of $\boldsymbol{\sigma}$, $\bar{\sigma}_i = \frac{\sigma_i}{\|\boldsymbol{\sigma}\|_1}$, and the convention that $\ln 0 = 0$ is adopted. This measure is maximized when all the singular values are equal, and minimized when the maximum singular value is much larger than other values.

2.3.2 Comparison

In the compression-aware method, it is common to employ sparsity regularizer on singular value vectors to encourage weight matrices to lie in a low rank subspace. The nuclear norm is the most frequently used. However, it simply makes everything closer to zero, which is unfriendly to keeping the energy of weight matrices. Hence, we prefer other measures that encourage the insignificant singular values (with small magnitude) to go to zero but keep the significant values (with large magnitude) or make them larger to maintain the energy. Hence, we choose Gini Index, Hoyer, and effective rank as potential objects, and make a comparison among them together with the nuclear norm.

We execute the comparison experiment on ResNet32 trained on the Cifar10 dataset. We utilize the most simple SVD to compress the network, and in the compression-aware training phase, we employ various sparsity measures on singular vector values of each weight matrix, with a hyperparameter λ to make the balance between accuracy and low rank. After this training, there are many singular values close to zero that can be set to zero without degrading performance. An appropriate indicator for identifying singular values retained was introduced in [98], namely spectral norm based indicator. This indicator is defined as the ratio of the largest discarded singular value to the maximal singular value. It is more efficient than the normal Frobenius norm based indicator [99], as

it can get rid of the interference caused by small and noisy singular values.

Fig. 7 shows the effect of the four sparsity measures. The most frequently used nuclear norm shows the worst performance. With the increase in the compression rate, the accuracy drops sharply. The reason behind this can be that at the time of pursuing a high compression ratio, the value of λ is increased, with more singular values imposed to zero. It dramatically destroys the expressive ability of the model. This figure also suggests that effective rank surpasses the rest measures for any compression regime. To be specific, when the accuracy is close to 85%, effective rank can achieve a compression ratio almost four times greater than the nuclear norm. And in the case of 90%, it can achieve two times greater than Hoyer. For a low compression regime, effective rank also has the greatest potential to achieve accuracy close to the original.

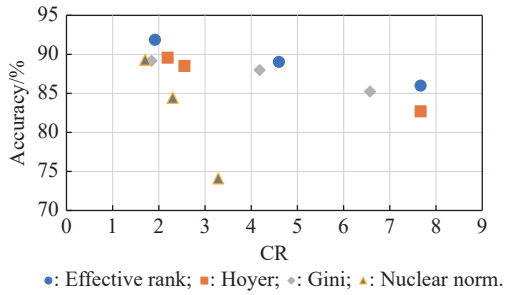

Fig. 7 Accuracy on Cifar10 v.s. compression ratio of the number of parameters in ResNet32

For the spectral norm based indicator, if we are in the need of discarding most of singular values to gain a high compression ratio, there are two choices: increase the value of maximum singular value or decrease the value of tiny singular values. However, increasing the value of the maximum singular value 10 times is much more difficult than decreasing the value of tiny singular values 10 times. Hence, we prefer a measure that can strongly encourage tiny singular values to reach 0. This is also the reason why effective rank can demonstrate great efficiency.

3. Integratable techniques

Apart from low rank approximation, there are other compression schemes that can result in a significant reduction of parameters at the expense of only a small drop in output accuracies, such as pruning [100], weight-sharing [101], sparsification [102] and knowledge distillation [20]. Undoubtedly, the integration of these parameter reduction techniques, namely parallel integration, can further enhance the efficiency of compression. While plenty of surveys suggest integrating various compression techniques, a detailed discussion on the combination between low rank approximation and other schemes is still lacking. In addition, not only the reduction of parameters but also the reduction of bits for representing parameters can significantly cut down the high complexity, which can be realized by quantization and entropy coding. Quantization can represent each parameter with lower bit-width, and entropy coding can use codewords to encode source symbols. Both techniques are orthogonal to the above parameter reduction methods. Hence, we can directly employ them on a compact model to gain a more compact representation, namely orthogonal integration. Table 6 lists representative works of different types of integration, and Table 7 lists whether these techniques can compress or accelerate models.

Table 6 Integratable techniques

Type of integration	Technique	Description	Representative integration works
Parallel integration	Pruning	Discard insignificant connections	[82, 98, 103]
	Sparsification	Zero out insignificant weights	[104−106]
	Weight sharing	Share weights across different connections	[107−109]
	Knowledge distillation	Transfer knowledge learned from teacher to student	[110−112]
Orthogonal integration	Quantization	Reduce precision	[113−115]
	Entropy coding	Encode weights into binary codewords	[116−118]

Table 7 Ability to compress and accelerate for various techniques

Technique	Acceleration	Compression
Pruning	√	√
Sparsification	√	√
Weight sharing	×	√
Knowledge distillation	√	√
Quantization	√	√
Entropy coding	×	√

3.1 Parallel integration

In this subsection, we give an all-round survey on how to integrate low rank approximation with other parallel compression techniques, including pruning, weight sharing, sparsification, and knowledge distillation. Through joint-way use, we can pursue a higher compression capacity.

3.1.1 Integration with pruning

Pruning is used to find unimportant connections in a full structure network and then abandon them, resulting in a compact structure without significant loss of accuracy. Pruning can be classified according to various levels of granularity, including weight-level, filter-level, and layer-level. Weight-level is the most flexible approach [102] and can gain the lowest memory costs by storing in sparse matrix format such as compressed sparse column (CSC) [20]. However, it leads to difficulty in inference due to the need for identifying each weight kept or abandoned. That is, this approach cannot speed up inference or save the memory footprint unless supported by hardware [119]. Layer-level aims at abandoning trivial layers, which is unsuitable for shallow networks [120]. To overcome these drawbacks, a more flexible and applicable approach, namely filter-level, is proposed [121]. Filter-level considers each filter as a unit and discards insignificant filters to obtain a compact model but with regular structures. Note that for two successive Conv layers, the removal of a filter in the first kernel leads to the removal of the input channel in the next kernel.

Filter pruning does not deal with the redundancy within a filter, while low rank approximation can overcome this by representing each filter in low rank format. Hence, it is promising to combine them to explore a higher compression ratio. Reference [122] proposed to perform filter pruning first and then employ Tucker decomposition on the pruned kernels. Experiments in [122] showed that the joint-way approach can achieve up to 57% higher compression ratio than either of them. Reference [98] exchanged the order of filter pruning and low-rank approximation since the smaller filters obtained by low rank approximation can reduce the probability of discarding essential filters. In addition, previous works pointed out that filter pruning is likely to prune more filters in deeper layers, resulting in still high computation costs of the whole network [123]. But with the help of low rank approximation, the shallow layers also can be compressed. Then, both high-level compression of memory and computation costs can be achieved.

One branch of works can achieve low rank approximation and filter pruning simultaneously via regularizers. In [82], the nuclear norm regularizer and the sparse group Lasso regularizer [124] were combined to make weight matrices not only low rank but also group sparse. Then the original layer can be represented by two smaller layers, followed by discarding insignificant input channels of the first layer and output channels of the second layer. Different from this method, [103] used one type of regularizer to achieve both two motivations. It represents a weight matrix by a basis matrix and a coefficient matrix. By imposing $\ell_{2,1}$ regularization both on the coefficient matrix and its transpose, the basis matrix can turn to be low rank and insignificant output channels are identified. Or, there are also some works that employ the two techniques on different modules of a network. For instance, aiming for Transformer architecture, [125] compressed the attention blocks by low rank approximation and applied to prune to feedforward blocks, which gains great enhancement.

3.1.2 Integration with sparsification

Sparsification in DNNs focuses on making weight matrices sparser so that sparse matrix computation can be employed to reduce high computation costs. Meanwhile, it can provide storage efficiency, as non-zeros and their locations can be recorded in compressed sparse row (CSR) [20] or ellpack sparse block (ESB) [126] format. There are two types of sparsification, namely irregular sparsity and structural sparsity. When the non-zeros are located randomly in the matrix, we call it irregular sparsity, which is flexible but may result in poor acceleration due to its irregular data access pattern. On the contrary, structural sparsity can achieve regular data access patterns. To be more specific, structural sparsity normally zeros out a series of continuous elements in the matrix.

Low rank approximation factors a matrix into smaller components, but these components still contain tiny elements which can be zeroed out without leading to a significant increase in approximation error. Hence, it is promising to combine low rank approximation and sparsification to achieve better compression. Sparse PCA (SPCA) [127] was a well-known instance to integrate factorization with sparsity. The main idea of SPCA is to make each principal component only contain a few features of data, so that SPCA is more explainable than PCA. There were also sparse HOSVD and sparse CP proposed in [65].

In [105], it has shown that surprisingly high sparsity can be achieved after two-stage decomposition. It was claimed that more than 90% of parameters can be zeroed out with less than 1% accuracy degradation on ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) dataset. In this algorithm, sparsity and low rank are achieved by employing ℓ_1 norm and $\ell_{2,1}$ norm respectively on a coefficient matrix. Finally, it converts the convolution operation in Conv layers into spare matrix multiplication, which dramatically reduces computation costs. Sparse SVD, i.e., factor matrices in SVD are sparsed, was proposed in [104], which outperforms truncated SVD. According to the view that a portion of the input and output neurons in a layer may be insignifi-

cant, the corresponding rows of the left and right singular matrix can be zeroed out. And considering the importance of entries in a row of left or right singular matrix decreases from left to right, the sparse SVD prefers to abandon entries nearing the right. The resulting structural sparsity allows basic linear algebra subprogram (BLAS) [128] libraries to be used for higher speed-up.

Aiming for RNNs, [106] proposed low-rank structured sparsity. Considering dimensional invariance in time, this method employs ℓ_1 regularization on the left and right singular matrix derived from SVD, resulting in a column-wise and row-wise sparse matrix without dimension distortion.

3.1.3 Integration with weight sharing

Weight sharing is defined as an operation that shares parameters across different connections in DNNs by exploiting redundancy. In order to design a more complex network with a better capacity for feature extraction, it is common to copy or reform some well-designed modules in a shallow network, and then add new modules to the end, yielding a deeper network. One typical network is the well-known ResNet [129]. Due to this similarity, it is promising to explore a more compact representation by sharing parameters across these similar subnetworks. For low rank approximation, similarly, the idea of sharing factor tensors across tensor decompositions of similar weight tensors can also be adopted.

A simple illustration of integration with weight sharing can be found in [108], where a set of 3D filter bases is shared across several or all convolutional layers. The search for bases is equivalent to low rank approximation of all the matrix-shaped kernels with a shared basis matrix.

Some tensor decomposition methods naturally combine weight sharing. For example, in the previously mentioned semi-tensor product-based tensor decomposition, STP can calculate a multiplication between a vector $\boldsymbol{x} \in \mathbf{R}^N$ and a weight vector $\boldsymbol{w} \in \mathbf{R}^P$, resulting an output vector $\boldsymbol{y} \in \mathbf{R}^{\frac{N}{P}}$. The $\dfrac{N}{P}$ entries in each block of $\boldsymbol{x}$ share one weight parameter of $\boldsymbol{W}$.

Alternatively, one branch of works shares factor tensors across tensor decompositions of weight tensors in different layers. Reference [107] proposed T-Basis, which constructs a set of third-order tensors. For an arbitrary-shaped tensor, each of its TR-cores can be represented as a linear combination of T-Basis. Hence, a compact representation of DNNs can be derived. Reference [109] proposed coupled TT, which contains a common component and an independent component. The common component is represented by shared TT-cores for similar network blocks, while the independent components in TT format are various from different layers to maintain the characteristics of each layer.

3.1.4 Integration with knowledge distillation

Knowledge distillation [130] is a promising solution, which aims to feed some extra knowledge learned from teacher networks (one or more complex networks) into a student network (much simpler network). With the help of a teacher, the student can achieve comparable accuracy but with much lower memory and computation costs compared with the teacher. Let $\boldsymbol{q}_s$ and $\boldsymbol{q}_t$ denote the softmax outputs of the student network and teacher network, respectively. The student network will be trained via aligning $\boldsymbol{q}_s$ and $\boldsymbol{q}_t$. But in the case that $\boldsymbol{q}_t$ is close to the one-hot code of true labels, the information contained in small values cannot be transferred to the student. Hence, a trick named temperature [130] is utilized to soften the distribution of both $\boldsymbol{q}_s$ and $\boldsymbol{q}_t$.

Networks compressed by low rank approximation is also a simpler network that can learn knowledge from the uncompressed version. In general, the decomposed networks are recovered by simply fine-tuning to minimize the cross-entropy function. However, the fine-tuning process always converges slowly and cannot recover the original accuracy well. Hence, this underlines the need for training the compressed network with information from the corresponding pre-training network.

However, it was demonstrated in [71] that it is difficult to train a student network deeper than the teacher network with knowledge distillation due to the undesirable phenomenon of vanishing gradient. Hence, a novel knowledge transfer (KT) was proposed in [111], which aligns both outputs and intermediate responses from a teacher (original) network to its student (compressed) network. Experiments show that it surpasses the common fine-tuning and knowledge distillation, particularly with a high compression ratio.

However, the KT method is still time-consuming and has a demand for a fully annotated large-scale training set, which may be infeasible in practice. Li et al. [110] proposed a revised knowledge distillation that only requires a few label-free samples. It adds a 1×1 Conv layer at the end of each block of the student network, and aligns block-level outputs of teacher and student by estimating the 1×1 Conv layer's parameters using least-squared regression. Since the number of parameters in 1×1 Conv layers is relatively small, only a few samples are necessary. It also enables fast model convergence, thereby saving much time for recovery of accuracy. After learning, the 1×1 Conv layer will be merged into the previous layer, without an increase in the number of parameters.

3.2 Orthogonal integration

3.2.1 Quantization

The operation that maps data from full precision to reduced precision is referred to as quantization. In the training and inference phase of DNNs, it is common to represent weights and activations in 32-bit. However, transferring data in 32-bit is a burden, and multiply-accumulate (MAC) will be operated between 32-bit floating-point values. In addition, energy consumed scales linearly to quadratically with the number of bits used. Hence, lowering the precision is necessary for the reduction of memory size, acceleration and energy saving.

There are some special advantages of applying quantization on neural networks. First, compared with continuous form, the discrete representations are more robust to noise [131,132] and are more similar to the way of storing information in human brains [133,134]. Second, both high generalization power [135,136] and high efficiency under limited resources [137] of discrete forms are actually what deep learning needs. Third, common compression methods, like low rank approximation, weight-sharing, and pruning, focus on either memory compression or acceleration so that it is deficient to achieve significant acceleration and compression simultaneously for a whole network, while quantization can conquer this challenge. In addition, it was shown in [138] that most of the weights and activations in DNNs are close to zero, which can greatly promote the compression ability of quantization. A more detailed survey about implementing quantization on DNNs could be found in [139,140].

A straight-forward way to combine low rank approximation and quantization is to consider the network compressed by tensor decomposition as a new network, which can be normally further compressed by various quantization methods. However, since there is already an approximation error derived from decomposition, the subsequent quantization will suffer from serious accuracy degradation. Hence, a novel integration method that considers low rank decomposition and quantization simultaneously instead of successively has the potential to address the challenge.

This idea can be found in [141], where both factors of Tucker format and activations are quantized, and with the help of knowledge distillation, the approximation error is minimized. In [114], quantization was introduced in PCA, where the component matrix and the coefficient matrix are quantized with different bit-widths. Together with a sparsity constraint on the coefficient matrix, the approximation error on the data manifold derived from low rank decomposition, sparsity and quantization will be minimized by an iterative projected gradient descent method.

Also, there are some approaches that directly extend basic tensor decomposition algorithms to tensor decompositions with quantized factors. For instance, quantized CP-alternating least squares (ALS) was proposed in [115], where each optimization iteration factors are quantized, and it is shown that the reconstruction error under ALS and quantized ALS are almost the same.

The above-mentioned methods are all aiming at approximating a tensor with quantized factors, which is not suitable for pre-set method. In [113], a quantized TT (QTT) was utilized for compressing three-dimensional convolutional neural networks. TT-cores in tensorizied neural networks are first quantized, and then the quantization of feedforward process is also made, achieving a three times faster inference than using only TT.

3.2.2 Entropy coding

Entropy coding is a lossless compression scheme, which encodes source symbols with a lower number of bits per symbol by exploiting the probability distribution of source [142]. Entropy coding originally adopted for data compression is introduced to further reduce the memory size of quantized DNNs by representing quantized weights with binary codewords [20]. It uses Huffman coding to further save 20% to 30% of network storage with no loss of accuracy.

Huffman coding is a theoretically optimal method to encode multivariate independence source symbols, but with the precondition that statistical characteristics of source symbols are already known. There is a problem with DNNs that statistical characteristics of weights calculated by histogram is a time-consuming preparation and are different for each network, even for a network fine-tuned. Hence, an encoding method without the need for exact statistics is more efficient for compressing DNNs.

One branch of works called universal coding, such as the variants of Lempel-ZivWelch [143−145] and the Burrows–Wheeler transform [146], can be applied to deal with this problem. The "universal" means that this coding method has a general probability model which can be slightly adapted to a broad class of input sources. In application, deep context-based adaptive binary arithmetic coder (DeepCABAC) [117], as a type of universal coding, is utilized to encode weights in DNNs. It is the first attempt to apply state-of-the-art video coding methods (e.g., CABAC) to DNNs. Compared with Huffman coding, DeepCABAC also has the advantage of higher efficiency in throughput.

However, both Huffman coding and DeepCABAC are fixed-to-variable (F2V) schemes in which the number of bits for each symbol is variable. Due to the variable length in codewords, it is inefficient for memory usage when decoding, and hence leads to high latency for inference. Instead, Tunstall coding [118], a variable-to-fixed

(V2F) method, is designed to fix the length of each codeword so that we can process multiple bits simultaneously and decode multiple encoded strings in parallel. It is reported that Tunstall coding can achieve around six times faster decoding than Huffman coding.

4. Low rank optimization for subspace training

4.1 Low rank function

For a differentiable real-valued function, if its gradient always lies in a fixed low-dimensional subspace, it can be called a low rank function [147]. The dimensionality of such subspaces is much lower than the number of independent variables, and it is referred to as the rank of the function. Ridge functions are the most common low rank function, which are defined as functions that can be converted into a univariate function by applying an affine transformation to the argument [148]. Hence, the gradient of such a function can also be projected into a line. For example, the least-square regression function which is a classic ridge function can be considered a rank-one function. The low rank property of ridge functions makes them widely used in classic statistics. They are utilized as regression functions in projection pursuit regression to deal with the curse of dimensionality and the noise in data [149]. In scientific computing, since the variables of functions for uncertainty quantification are always correlated, the concept of active subspaces can be utilized to reveal a set of independent variables whose fluctuation can lead to the most significant change [150,151].

Low rank property has also been found in the training phase of DNNs. In DNNs, the number of trainable parameters is always far more than that of training samples. Thus, for this type of over-parameterized model, it is possible to guess that there is a large part of the parameters that will remain unchanged during the whole training phase. More generally, there is a hypothesis that the training trajectory of parameters lies in a subspace constructed by a few irrelevant variables. That is to say, the optimization of millions of parameters can be equivalent to optimization in a tiny subspace. There is also evidence that the gradient of various DNNs will gradually remain in a tiny subspace spanned by a few top eigenvectors of the Hessian [152].

4.2 Subspace training

In deep learning, the challenge that the process of training converges very slowly is a thorny obstacle. The slow convergence is caused by the dominating first-order method, i.e., gradient descent-based methods. This problem can be relieved by second-order methods which utilize the information derived from Hessian matrices. Moreover, the second-order method is not sensitive to the learning rate, so no specific learning rate schedule needs to be designed. However, due to the massive parameters in DNNs, it is a computational burden to calculate Hessian matrices. Some approaches such as Adam [153], RMSprop [154], and AdaGrad [155] utilize part of second-order information, like momentum and accumulation information, have already surpassed the performance of conventional gradient-based methods.

In order to apply second-order methods such as quasi-Newton method [156] to network training, the straightforward way is to reduce the number of parameters that need to be optimized. In view of the low rank structure discovered in DNNs, it is promising to optimize the whole network in a subspace using quasi-Newton method, without the loss of accuracy. DLDR-based Quasi-Newton method [157] is introduced to save 35% of training time versus SGD [24]. To be specific, in this algorithm, dynamic linear dimensionality reduction (DLDR) is devised to identify the low-dimensional subspace constructed in some important directions which can contribute significantly to the variance of the loss function. It achieves this by sampling the training trajectory and then performing PCA to analyse the dominating directions. Then, second-order optimization can be directly executed in this tiny subspace, resulting in fast convergence.

4.3 Spatial redundancy and temporal redundancy

While model compression exploits the redundancy in networks to reduce memory and computation complexity, subspace training exploits the redundancy to reduce training time. In other words, the objective of model compression and subspace training is spatial efficiency and temporal efficiency, respectively. Since they both exploit redundancy, we are wondering whether the redundancy they deal with is of the same origin or not.

We analyse this by performing subspace training on low rank approximated networks to determine if subspace training has a poor performance on compressed networks. If so, it is evidence that the redundancy decreased by model compression is insufficient for subspace training, i.e., the low rank property in time domain disappears.

Here, we perform a simple experiment on LetNet-300-100 with Mixed National Institute of Standards and Technology (MNIST) dataset. LeNet-300-100 contains two hidden fully connected layers with output dimensions 300 and 100, and an output layer with dimension 10. We apply SVD on the first two layers and then fine-tune. We record the training trajectory and establish a 5D subspace by performing PCA. To see if such a tiny subspace is suf-

ficient, we project weights onto this subspace and calculate the normalized approximate error. Fig. 8 shows that as the rank decreases, the normalized error increases almost linearly. It suggests that the higher the compression ratio, the less suitable the subspace with such low fixed dimensionality is. In other words, model compression decreases the redundancy subspace training can exploit. The normalized error is the ratio of ℓ_2 norm of error between original parameters and projected parameters and ℓ_2 norm of the original parameters. The rank is in respect to SVD. "Base" is the uncompressed network.

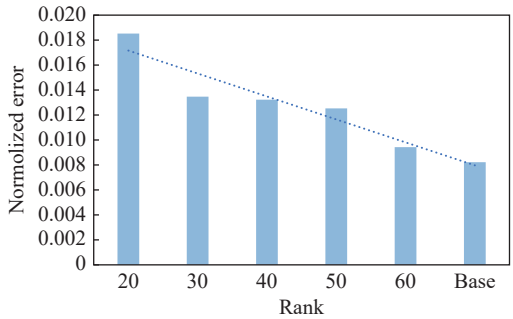

Fig. 8 Normalized error v.s. rank when projecting SVD-compressed LeNet-300-100 on a 5D subspace

Also, we can figure that after low rank decomposition, a higher-dimensional subspace is in need. As shown in Fig. 9, increasing the dimensionality of subspace has a greater effect on the highly compressed network. the dimensionality of subspace ranges from 5 to 15. Under all the rank settings, normalized error goes to zero when the dimensionality is equal to 12. But there is a sharp descent when the dimensionality is increased from 11 to 12 for rank=20. That is to say, a slight drop in dimensionality is serious for a highly compressed network. When a network is compressed extremely, there is little redundancy in time domain.

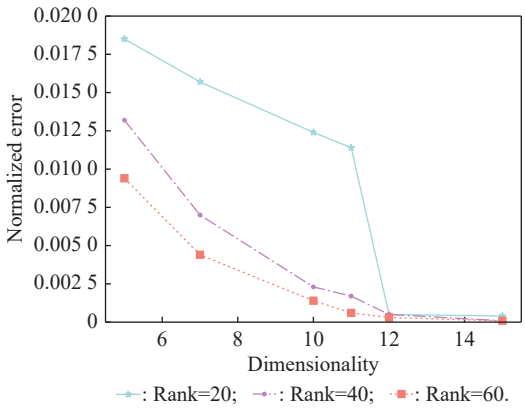

Fig. 9 Normalized error v.s. dimensionality of subspace under different ranks (different compression extents)

4.4 Making a balance

Since redundancy exploited by model compression and subspace training are of the same origin, there is a balance between spatial efficiency and temporal efficiency. If we assign most of the redundancy to model compression, we can obtain a compact network and hence achieve spatial efficiency, but little redundancy is left for subspace training. Conversely, if we are in need to train a network quickly, we should promise to assign most of the redundancy to subspace training.

For model compression, the training of a tensorized neural network (TNN) is much time-consuming than that of the original network. Hence, there is a need for utilizing subspace training to accelerate the training of TNN. Intuitively, for a highly compressed TNN, since there is little redundancy, it is inefficient to train such a TNN in a tiny subspace. Fig. 10 shows the performance of subspace training when applied to TT-based TNNs with various compression regimes. The base network is ResNet32 trained on Cifar10 dataset. All the experiments run 15 epochs (saving 35% time of SGD method) with Quasi-Newton method and the subspace is fixed to 40D. In this figure, the orange line (the case in which TT-Net is trained in normal way) is almost a horizontal line, but the green line (trained in subspace) descends sharply at the time of high compression ratio. It suggests that subspace training can be combined with model compression to achieve spatio-temporal efficiency under a moderate compression regime, but such a tiny space is not suitable for an extremely compressed network.

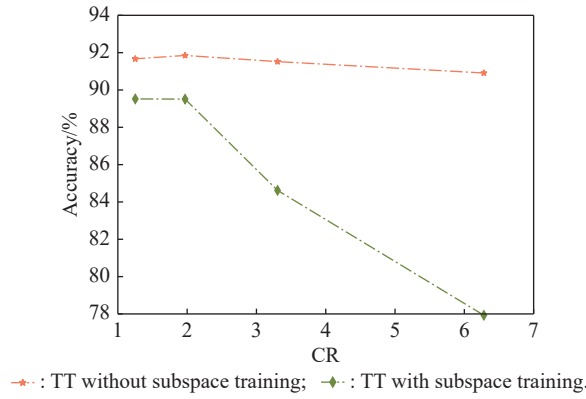

Fig. 10 Comparison of the accuracy degradation when applying subspace training to TT-Nets

Hence, under an extreme compression regime, it is essential to increase the dimensionality of subspace to relieve the accuracy degradation. But it is infeasible to increase dimensionality blindly, as the number of sampling epochs will also increase, i.e., lessen temporal efficiency. Fig. 11 shows the effect of increasing the dimen-

sionality of subspace for a highly compressed TT-Net. In Fig. 11, the dashed line represents the accuracy of teaining TT-Nets in an normal way. It demonstrates that as the dimensionality of subspace increases, the accuracy degradation of subspace training decreases. When the dimensionality is increased to 55, we can achieve a good accuracy close to the original, but it is worth noting that the total time (time for subspace training and for sampling) is near the normal training time. However, in the case that we want to train a compact TNN quickly and a small drop in accuracy can be tolerated, it is a good choice to train such a network in a moderate subspace.

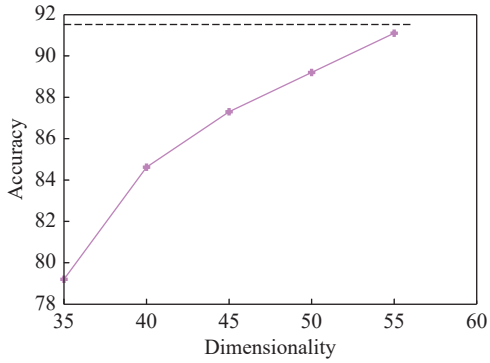

Fig. 11 Effect of increasing the dimensionality of subspace on training TT-Nets in subspace

5. Conclusions and future directions

In this paper, two types of low rank tensor optimization for efficient deep learning are discussed, namely low rank approximation for model compression and subspace training for fast convergence. For low rank approximation, we list various efficient tensor decomposition methods and introduce three types of optimization methods. Since sparsity measure is applied frequently in low rank approximation, we make a comparison among common measures, and experiments show that effective rank can achieve the best accuracy-compression tradeoff. In addition, we investigate how to integrate low rank approximation with other compression techniques. Then, we give a brief introduction to subspace training and analyze that redundancy exploited by subspace training and low rank approximation is of the same origin. Further, we make a discussion on how to combine the two to accelerate the training of tensorized neural networks.

However, up to now, few works focus on integrating more than three types of parameter reduction compression techniques, which is more promising to take maximum advantage of redundancy in networks. Further, it is possible to devise a flexible framework to integrate all kinds of compression techniques.

In practice, low computation complexity is not equivalent to low latency [158], and the energy consumed by computation is only a small part of the total energy for inference [159,160]. But most works take FLOPs and memory size as benchmarks. That is to say, an advanced algorithm with very low complexity may not be applied to battery-powered mobile devices. Hence, more efforts are needed in decreasing the energy consumption of DNNs.

For subspace training, the temporal efficiency is still limited, as the quasi-Newton method is still based on the gradient of the original millions of parameters. Direct optimization on several independent variables is still to be studied. In addition, since the sampling procedure occupies most of the training time, there is a need to introduce new techniques to construct subspace with fewer sample epochs. One potential way is to represent all the parameters in tensor format and apply tensor decomposition to better analyze principal components, i.e., higher-order PCA [68].

References

[1] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 2017, 60(6): 84–90.

[2] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition. https://arxiv.org/abs/1409.1556.

[3] JIANG Y G, WU Z X, WANG J, et al. Exploiting feature and class relationships in video categorization with regularized deep neural networks. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2017, 40(2): 352–364.

[4] ZHANG Z H, LIU Y P, CAO X Y, et al. Scalable deep compressive sensing. https://arxiv.org/abs/2101.08024.

[5] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need. https://arxiv.org/abs/1706.03762.

[6] GRAVES A, MOHAMED A R, HINTON G. Speech recognition with deep recurrent neural networks. Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2013: 6645−6649.

[7] HINTON G E, SRIVASTAVA N, KRIZHEVSKY A, et al. Improving neural networks by preventing coadaptation of feature detectors. https://arxiv.org/abs/1207.0580.

[8] DENIL M, SHAKIBI B, DINH L, et al. Predicting parameters in deep learning. Proc. of the 26th International Conference on Neural Information Processing Systems, 2023, 12(2): 2148–2156.

[9] KIM Y D, PARK E, YOO S, et al. Compression of deep convolutional neural networks for fast and low power mobile applications. https://arxiv.org/abs/1511.06530.

[10] LANE N D, BHATTACHARYA S, GEORGIEV P, et al. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. Proc. of the International Workshop on Internet of Things Towards Applications, 2015: 7−12.

[11] ABDUL HAMID N, MOHD NAWI N, GHAZALI R, et al. Accelerating learning performance of back propagation

algorithm by using adaptive gain together with adaptive momentum and adaptive learning rate on classification problems. Proc. of the International Conference on Ubiquitous Computing and Multimedia Applications, 2011: 559−570.

[12] LEBEDEV V, GANIN Y, RAKHUBA M, et al. Speeding-up convolutional neural networks using finetuned CP-decomposition. https://arxiv.org/abs/1412.6553v2

[13] JADERBERG M, VEDALDI A, ZISSERMAN A. Speeding up convolutional neural networks with low rank expansions. https://arxiv.org/abs/1405.3866.

[14] WANG W Q, SUN Y F, ERIKSSON B, et al. Wide compression: tensor ring nets. Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, 2018: 9329−9338.

[15] LIU Y P, LIU J N, LONG Z, et al. Tensor decomposition in deep networks. Tensor Computation for Data Analysis. Cham: Springer, 2022.

[16] LUO J H, WU J X, LIN W Y. Thinet: a filter level pruning method for deep neural network compression. Proc. of the IEEE International Conference on Computer Vision, 2017: 5058−5066.

[17] ZHANG T Y, YE S K, ZHANG K Q, et al. A systematic dnn weight pruning framework using alternating direction method of multipliers. Proc. of the European Conference on Computer Vision, 2018: 184−199.

[18] ULLRICH K, MEEDS E, WELLING M. Soft weight sharing for neural network compression. https://arxiv.org/abs/1702.04008.

[19] HUANG J Z, ZHANG T, METAXAS D. Learning with structured sparsity. Journal of Machine Learning Research, 2011, 12(103): 3371–3412.

[20] HAN S, MAO H Z, DALLY W J. Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. https://arxiv.org/abs/1510.00149v4.

[21] GONG Y C, LIU L, YANG M, et al. Compressing deep convolutional networks using vector quantization. https://arxiv.org/abs/1412.6115.

[22] WU J X, LENG C, WANG Y H, et al. Quantized convolutional neural networks for mobile devices. Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, 2016: 4820−4828.

[23] WANG M L, PAN Y, YANG X L, et al. Tensor networks meet neural networks: a survey. https://arxiv.org/abs/2302.09019.

[24] RUDER S. An overview of gradient descent optimization algorithms. https://arxiv.org/abs/1609.04747.

[25] DENG L, LI G Q, HAN S, et al. Model compression and hardware acceleration for neural networks: a comprehensive survey. Proceedings of the IEEE, 2020, 108(4): 485–532.

[26] CHOUDHARY T, MISHRA V, GOSWAMI A, et al. A comprehensive survey on model compression and acceleration. Artificial Intelligence Review, 2020, 53(7): 5113–5155.

[27] LIU J N, ZHU C, LONG Z, et al. Tensor regression. https://arxiv.org/abs/2308.11419.

[28] LIU Y P. Tensors for data processing: theory methods and applications. San Diego: Elsevier Science & Technology, 2021.

[29] FENG L L, ZHU C, LONG Z, et al. Multiplex transformed tensor decomposition for multidimensional image recovery. IEEE Trans. on Image Processing, 2023, 32: 3397−3412.

[30] ZHANG X Y, ZOU J H, HE K M, et al. Accelerating very deep convolutional networks for classification and detection. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2015, 38(10): 1943–1955.

[31] TUCKER L R. Implications of factor analysis of three-way matrices for measurement of change. Problems in Measuring Change, 1963, 15: 122–137.

[32] GRASEDYCK L. Hierarchical singular value decomposition of tensors. Society for Industrial and Applied Mathematics, 2010, 31(4): 2029–2054.

[33] OSELEDETS I V. Tensor-train decomposition. Siam Journal on Scientific Computing, 2011, 33(5): 2295–2317.

[34] ZHAO Q B, ZHOU G X, XIE S L, et al. Tensor ring decomposition. https://arxiv.org/abs/1606.05535.

[35] DE LATHAUWER L. Decompositions of a higher-order tensor in block terms—part II: definitions and uniqueness. SIAM Journal on Matrix Analysis and Applications, 2008, 30(3). DOI: 10.1137/070690729.

[36] HAMEED M G A, TAHAEI M S, MOSLEH A, et al. Convolutional neural network compression through generalized kronecker product decomposition. Proc. of the AAAI Conference on Artificial Intelligence, 2022: 771−779.

[37] ZHAO H L, LIU Y P, HUANG X L, et al. Semi-tensor product-based tensor decomposition for neural network compression. https://arxiv.org/abs/2109.15200.

[38] GARIPOV T, PODOPRIKHIN D, NOVIKOV A, et al. Ultimate tensorization: compressing convolutional and fc layers alike. https://arxiv.org/abs/1611.03214.

[39] YE J M, LI G X, CHEN D, et al. Block-term tensor neural networks. Neural Networks, 2020, 130: 11–21.

[40] WU B J, WANG D H, ZHAO G S, et al. Hybrid tensor decomposition in neural network compression. Neural Networks, 2020, 132: 309–320.

[41] LIU Y P, LONG Z, HUANG H Y, et al. Low CP rank and tucker rank tensor completion for estimating missing components in image data. IEEE Trans. on Circuits and Systems for Video Technology, 2019, 30(4): 944–954.

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

[43] LIU Y P, LONG Z, ZHU C. Image completion using low tensor tree rank and total variation minimization. IEEE Trans. on Multimedia, 2018, 21(2): 338–350.

[44] LIU Y P, LIU J N, ZHU C. Low-rank tensor train coefficient array estimation for tensor-on-tensor regression. IEEE Trans. on Neural Networks and Learning Systems, 2020, 31(12): 5402–5411.

[45] NOVIKOV A, PODOPRIKHIN D, OSOKIN A, et al. Tensorizing neural networks. https://arxiv.org/abs/1509.06569.

[46] YIN M, SUI Y, YANG W Z, et al. HODEC: towards efficient high-order decomposed convolutional neural networks. Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022: 12299−12308.

[47] HUANG H Y, LIU Y P, LONG Z, et al. Robust low rank tensor ring completion. IEEE Trans. on Computational Imaging, 2020, 6: 1117–1126.

[48] LIU J N, ZHU C, LIU Y P. Smooth compact tensor ring regression. IEEE Trans. on Knowledge and Data Engineering, 2020, 34(9): 4439–4452.

[49] LONG Z, ZHU C, LIU J N, et al. Bayesian low rank tensor ring for image recovery. IEEE Trans. on Image Processing, 2021, 30: 3568–3580.

[50] THAKKER U, BEU J, GOPE D, et al. Compressing RNNs

for IoT devices by 15−38x using Kronecker products. https://arxiv.org/abs/1906.02876.

[51] CHENG D Z, QI H S, XUE A C. A survey on semi-tensor product of matrices. Journal of Systems Science and Complexity, 2007, 20(2): 304–322.

[52] LIEBENWEIN L, MAALOUF A, FELDMAN D, et al. Compressing neural networks: towards determining the optimal layer-wise decomposition. Advances in Neural Information Processing Systems, 2021, 34: 5328–5344.

[53] IDELBAYEV Y, CARREIRA-PERPINÁN M A. Low-rank compression of neural nets: learning the rank of each layer. Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020: 8049−8059.

[54] YIN M, SUI Y, LIAO S Y, et al. Towards efficient tensor decomposition-based dnn model compression with optimization framework. Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021: 10674−10683.

[55] YIN M, PHAN H, ZANG X, et al. BATUDE: budget-aware neural network compression based on tucker decomposition. Proc. of the AAAI Conference on Artificial Intelligence, 36(8): 8874−8882.

[56] NAKAJIMA S, SUGIYAMA M, BABACAN S D, et al. Global analytic solution of fully-observed variational bayesian matrix factorization. The Journal of Machine Learning Research, 2013, 14(1): 1–37.

[57] REEVES C R. Modern heuristic techniques for combinatorial problems. New York: John Wiley & Sons, 1993.

[58] CHENG Z Y, LI B P, FAN Y W, et al. A novel rank selection scheme in tensor ring decomposition based on reinforcement learning for deep neural networks. Proc. of the ICASSP IEEE International Conference on Acoustics, 2020: 3292−3296.

[59] SAMRAGH M, JAVAHERIPI M, KOUSHANFAR F. AutoRank: automated rank selection for effffective neural network customization. Proc. of the ML-for Systems Workshop at the 46th International Symposium on Computer Architecture, 2019. DOI: 10.1109/JETCAS.2021.3127433.

[60] MITCHELL B C, BURDICK D S. Slowly converging parafac sequences: swamps and two-factor degeneracies. Journal of Chemometrics, 1994, 8(2): 155–168.

[61] HARSHMAN R A. The problem and nature of degenerate solutions or decompositions of 3-way arrays. https://www.psychology.uwo.ca/faculty/harshman/aim2004.pdf.

[62] KRIJNEN W P, DIJKSTRA T K, STEGEMAN A. On the non-existence of optimal solutions and the occur rence of degeneracy in the candecomp/parafac model. Psychometrika, 2008, 73(3): 431–439.

[63] DENTON E L, ZAREMBA W, BRUNA J, et al. Exploiting linear structure within convolutional networks for efficient evaluation. https://arxiv.org/abs/1404.0736.

[64] ASTRID M, LEE S I. CP-decomposition with tensor power method for convolutional neural networks compression. Proc. of the IEEE International Conference on Big Data and Smart Computing, 2017: 115−118.

[65] ALLEN G. Sparse higher-order principal components analysis. Proc. of the Artifificial Intelligence and Statistics, 2012: 27−36.

[66] PHAN A H, SOBOLEV K, SOZYKIN K, et al. Stable low-rank tensor decomposition for compression of con volutional neural network. Proc. of the European Conference on Computer Vision, 2020: 522−539.

[67] VEERAMACHENENI L, WOLTER M, KLEIN R, et al. Canonical convolutional neural networks. https://arxiv.org/abs/2206.01509v1.

[68] KOLDA T G, BADER B W. Tensor decompositions and applications. SIAM Review, 2009, 51(3): 455–500.

[69] ESPIG M, HACKBUSCH W, HANDSCHUH S, et al. Optimization problems in contracted tensor networks. Computing and Visualization in Science, 2011, 14(6): 271–285.

[70] PHAN A H, SOBOLEV K, ERMILOV D, et al. How to train unstable looped tensor network. https://arxiv.org/abs/2203.02617.

[71] GLOROT X, BENGIO Y. Understanding the difficulty of training deep feed forward neural networks. Journal of Machine Learning Research, 2010: 249−256.

[72] PAN Y, SU Z Y, LIU A, et al. A unified weight initialization paradigm for tensorial convolutional neural networks. Proc. of the International Conference on Machine Learning, 2022: 17238−17257.

[73] ZOPH B, LE Q V. Neural architecture search with reinforcement learning. https://arxiv.org/abs/1611.01578.

[74] LI N N, PAN Y, CHEN Y R, et al. Heuristic rank selection with progressively searching tensor ring network. Complex & Intelligent Systems, 2022, 8(2): 771–785.

[75] DEB K, PRATAP A, AGARWAL S, et al. A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002, 6(2): 182–197.

[76] HAWKINS C, ZHANG Z. Bayesian tensorized neural networks with automatic rank selection. Neurocomputing, 2021, 453: 172–180.

[77] RAI P, WANG Y J, GUO S B, et al. Scalable bayesian low-rank decomposition of incomplete multiway tensors. Proc. of the International Conference on Machine Learning, 2014: 1800−1808.

[78] GUHANIYOGI R, QAMAR S, DUNSON D B. Bayesian tensor regression. The Journal of Machine Learning Research, 2017, 18(1): 2733–2763.

[79] BAZERQUE J A, MATEOS G, GIANNAKIS G B. Rank regularization and bayesian inference for tensor completion and extrapolation. IEEE Trans. on Signal Processing, 2013, 61(22): 5689–5703.

[80] EO M, KANG S, RHEE W. An effective low-rank compression with a joint rank selection followed by a compression-friendly training. Neural Networks, 2023, 161: 165–177.

[81] CAI J F, CANDÈS E J, SHEN Z W. A singular value thresholding algorithm for matrix completion. https://arxiv.org/abs/0810.3286.

[82] ALVAREZ J M, SALZMANN M. Compression-aware training of deep networks. https://arxiv.org/abs/1711.02638.

[83] XU Y H, LI Y X, ZHANG S, et al. Trained rank pruning for efficient deep neural networks. Proc. of the 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition, 2019: 14−17.

[84] AVRON H, KALE S, KASIVISWANATHAN S, et al. Efficient and practical stochastic subgradient descent for nuclear norm regularization. https://arxiv.org/abs/1206.6384.

[85] YANG H R, TANG M X, WEN W, et al. Learning low-rank deep neural networks via singular vector orthogonality regularization and singular value sparsification. Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2020: 678−679.

[86] CARREIRA-PERPINÁN M A, IDELBAYEV Y. Learning-compression algorithms for neural net pruning. Proc. of the

IEEE Conference on Computer Vision and Pattern Recognition, 2018: 8532−8541.
[87] ZIMMER M, SPIEGEL C, POKUTTA S. Compression aware training of neural networks using Frank-Wolfe. https://arxiv.org/abs/2205.11921.
[88] SHI L, HUANG X L, FENG Y L, et al. Sparse kernel regression with coefficient-based ℓ_q-regularization. Journal of Machine Learning Research, 2019, 20(161): 1–44.
[89] XU P, TIAN Y, CHEN H F, et al. ℓ_p norm iterative sparse solution for EEG source localization. IEEE Trans. on Biomedical Engineering, 2007, 54(3): 400–409.
[90] BOGDAN M, BERG E V D, SU W, et al. Statistical estimation and testing via the sorted ℓ_1 norm. https://arxiv.org/abs/1310.1969.
[91] HUANG X L, LIU Y P, SHI L, et al. Two-level ℓ_1 minimization for compressed sensing. Signal Processing, 2015, 108: 459–475.
[92] DALTON H. The measurement of the inequality of incomes. The Economic Journal, 1920, 30(119): 348–361.
[93] LORENZ M O. Methods of measuring the concentration of wealth. Publications of the American statistical association, 1905, 9(70): 209–219.
[94] RICKARD S. Sparse sources are separated sources. Proc. of the 14th European signal processing conference, 2006: 1−5.
[95] HURLEY N, RICKARD S, CURRAN P. Parameterized lifting for sparse signal representations using the gini index. Proc. of the Signal Processing with Adaptative Sparse Structured Representations Conference, 2005. http://spars05.irisoa.fr/ACTES/TS4-4.pdf.
[96] HOYER P O. Non-negative matrix factorization with sparseness constraints. Journal of Machine Learning Research, 2004, 5(9): 1457–1469.
[97] ROY O, VETTERLI M. The effective rank: a measure of effective dimensionality. Proc. of the 15th European signal processing conference, 2007: 606−610.
[98] CHEN Z, CHEN Z B, LIN J X, et al. Deep neural network acceleration based on low-rank approximated channel pruning. IEEE Trans. on Circuits and Systems I: Regular Papers, 2020, 67(4): 1232–1244.
[99] OSAWA K, YOKOTA R. Evaluating the compression efficiency of the filters in convolutional neural networks. Proc. of the International Conference on Artificial Neural Networks, 2017: 459−466.
[100] BLALOCK D, GONZALEZ ORTIZ J J, FRANKLE J, et al. What is the state of neural network pruning? https://arxiv.org/abs/2003.03033.
[101] CHEN W L, WILSON J, TYREE S, et al. Compressing neural networks with the hashing trick. Proc. of the International Conference on Machine Learning, 2015: 2285−2294.
[102] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural network. Proc. of the 28th International Conference on Neural Information Processing Systems, 2015, 1: 1135–1143.
[103] RUAN X F, LIU Y F, YUAN C F, et al. EDP: an efficient decomposition and pruning scheme for convolutional neural network compression. IEEE Trans. on Neural Networks and Learning Systems, 2020, 32(10): 4499–4513.
[104] SWAMINATHAN S, GARG D, KANNAN R, et al. Sparse low rank factorization for deep neural network compression. Neurocomputing, 2020, 398: 185−196.
[105] LIU B Y, WANG M, FOROOSH H, et al. Sparse convolutional neural networks. Proc. of the IEEE Conference on Computer Vision and Pattern Recognition. 2015: 806−814.
[106] WEN W J, YANG F, SU Y F, et al. Learning low-rank structured sparsity in recurrent neural networks. Proc. of the IEEE International Symposium on Circuits and Systems, 2020. DOI: 10.1109/ISCAS45731.2020.9181239.
[107] OBUKHOV A, RAKHUBA M, GEORGOULIS S, et al. T-basis: a compact representation for neural networks. Proc. of the International Conference on Machine Learning, 2020: 7392−7404.
[108] LI Y W, GU S H, GOOL L V, et al. Learning filter basis for convolutional neural network compression. Proc. of the IEEE/CVF International Conference on Computer Vision, 2019: 5623−5632.
[109] SUN W Z, CHEN S W, HUANG L, et al. Deep convolutional neural network compression via coupled tensor decomposition. IEEE Journal of Selected Topics in Signal Processing, 2020, 15(3): 603–616.
[110] LI T H, LI J G, LIU Z, et al. Few sample knowledge distillation for efficient network compression. Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020: 14639−14647.
[111] LIN S H, JI R R, CHEN C, et al. Holistic CNN compression via low-rank decomposition with knowledge transfer. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2018, 41(12): 2889–2905.
[112] SADHUKHAN R, SAHA A, MUKHOPADHYAY J, et al. Knowledge distillation inspired fine-tuning of tucker decomposed cnns and adversarial robustness analysis. Proc. of the IEEE International Conference on Image Processing, 2020: 1876−1880.
[113] LEE D, WANG D H, YANG Y K, et al. QTTNET: quantized tensor train neural networks for 3D object and video recognition. Neural Networks, 2021, 141: 420–432.
[114] KUZMIN A, VAN BAALEN M, NAGEL M, et al. Quantized sparse weight decomposition for neural network compression. https://arxiv.org/abs/2207.11048v1.
[115] NEKOOEI A, SAFARI S. Compression of deep neural networks based on quantized tensor decomposition to implement on reconfigurable hardware platforms. Neural Networks, 2022, 150: 350–363.
[116] CHOI Y, EL-KHAMY M, LEE J. Universal deep neural network compression. IEEE Journal of Selected Topics in Signal Processing, 2020, 14(4): 715–726.
[117] WIEDEMANN S, KIRCHHOFFER H, MATLAGE S, et al. Deepcabac: a universal compression algorithm for deep neural networks. IEEE Journal of Selected Topics in Signal Processing, 2020, 14(4): 700–714.
[118] CHEN C Y, WANG Z, CHEN X W, et al. Efficient tunstall decoder for deep neural network compression. Proc. of the 58th ACM/IEEE Design Automation Conference, 2021: 1021−1026.
[119] HAN S, LIU X Y, MAO H Z, et al. EIE: efficient inference engine on compressed deep neural network. Proc. of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture. DOI: 10.1109/ISCA.2016.30.
[120] CHEN S, ZHAO Q. Shallowing deep networks: Layerwise pruning based on feature representations. IEEE Trans. on pattern analysis and machine intelligence, 2018, 41(12): 3048–3056.
[121] HUANG Q G, ZHOU K, YOU S, et al. Learning to prune filters in convolutional neural networks. Proc. of the IEEE Winter Conference on Applications of Computer Vision, 2018: 709−718.
[122] GOYAL S, CHOUDHURY A R, SHARMA V. Compres-

sion of deep neural networks by combining pruning and low rank decomposition. Proc. of the IEEE International Parallel and Distributed Processing Symposium Workshops, 2019: 952−958.
[123] MOLCHANOV P, TYREE S, KARRAS T, et al. Pruning convolutional neural networks for resource efficient inference. https://arxiv.org/abs/1611.06440.
[124] ALVAREZ J M, SALZMANN M. Learning the number of neurons in deep networks. https://arxiv.org/abs/1611.06321v1.
[125] KUMAR A. Vision transformer compression with structured pruning and low rank approximation. https://arxiv.org/abs/2203.13444.
[126] LIU X, SMELYANSKIY M, CHOW E, et al. Efficient sparse matrix-vector multiplication on x86-based manycore processors. Proc. of the 27th International ACM Conference on on Supercomputing, 2013: 273−282.
[127] ZOU H, HASTIE T, TIBSHIRANI R. Sparse principal component analysis. Journal of computational and graphical statistics, 2006, 15(2): 265–286.
[128] LEBEDEV V, LEMPITSKY V. Fast convnets using group-wise brain damage. Proc. of the IEEE Conference on Computer Vision and Pattern Recognition, 2016: 2554−2564.
[129] LECUN Y, BENGIO Y, HINTON G. Deep learning. Nature, 2015, 521(7553): 436–444.
[130] HINTON G, VINYALS O, DEAN J, et al. Distilling the knowledge in a neural network. https://arxiv.org/abs/1503.02531.
[131] CHAUDHURI R, FIETE I. Computational principles of memory. Nature Neuroscience, 2016, 19(3): 394–403.
[132] FAISAL A A, SELEN L P, WOLPERT D M. Noise in the nervous system. Nature Reviews Neuroscience, 2008, 9(4): 292–303.
[133] VANRULLEN R, KOCH C. Is perception discrete or continuous? Trends in Cognitive Sciences, 2003, 7(5): 207−213.
[134] TEE J, TAYLOR D P. Is information in the brain represented in continuous or discrete form? IEEE Trans. on Molecular, Biological and Multi-Scale Communications, 2020, 6(3): 199–209.
[135] KHAW M W, STEVENS L, WOODFORD M. Discrete adjustment to a changing environment: experimental evidence. Journal of Monetary Economics, 2017, 91: 88–103.
[136] LATIMER K W, YATES J L, MEISTER M L, et al. Single-trial spike trains in parietal cortex reveal discrete steps during decision-making. Science, 2015, 349(6244): 184–187.
[137] VARSHNEY L R, SJÖSTRÖM P J, CHKLOVSKII D B. Optimal information storage in noisy synapses under resource constraints. Neuron, 2006, 52(3): 409–423.
[138] LIN D, TALATHI S, ANNAPUREDDY S. Fixed point quantization of deep convolutional networks. Proc. of the International Conference on Machine Learning, 2016: 2849−2858.
[139] GHOLAMI A, KIM S, DONG Z, et al. A survey of quantization methods for efficient neural network inference. https://arxiv.org/abs/2103.13630.
[140] NAGEL M, FOURNARAKIS M, AMJAD R A, et al. A white paper on neural network quantization. https://arxiv.org/abs/2106.08295.
[141] KOZYRSKIY N, PHAN A H. CNN acceleration by lowrank approximation with quantized factors. https://arxiv.org/abs/2006.08878.
[142] RECANATESI S, FARRELL M, ADVANI M, et al. Dimensionality compression and expansion in deep neural networks. https://arxiv.org/abs/1906.00443v1.
[143] ZIV J, LEMPEL A. A universal algorithm for sequential data compression. IEEE Trans. on information theory, 1977, 23(3): 337–343.
[144] ZIV J, LEMPEL A. Compression of individual sequences via variable-rate coding. IEEE Trans. on Information Theory, 1978, 24(5): 530–536.
[145] WELCH T A. A technique for high-performance data compression. Computer, 1984, 17(6): 8–19.
[146] EFFROS M, VISWESWARIAH K, KULKARNI S R, et al. Universal lossless source coding with the burrows wheeler transform. IEEE Trans. on Information Theory, 2002, 48(5): 1061–1081.
[147] COSSON R, JADBABAIE A, MAKUR A, et al. Gradient descent for low-rank functions. https://arxiv.org/abs/2206.08257.
[148] LOGAN B F, SHEPP L A. Optimal reconstruction of a function from its projections. Duke Mathematical Journal, 1975, 42(4): 645–659.
[149] DONOHO D L, JOHNSTONE I M. Projection-based approximation and a duality with kernel methods. The Annals of Statistics, 1989: 58−106.
[150] CONSTANTINE P G, EMORY M, LARSSON J, et al. Exploiting active subspaces to quantify uncertainty in the numerical simulation of the HyShot II scramjet. Journal of Computational Physics, 2015, 302: 1–20.
[151] LIU Y P, DE VOS M, GLIGORIJEVIC I, et al. Multistructural signal recovery for biomedical compressive sensing. IEEE Trans. on Biomedical Engineering, 2013, 60(10): 2794–2805.
[152] GUR-ARI G, ROBERTS D A, DYER E. Gradient descent happens in a tiny subspace. https://arxiv.org/abs/1812.04754.
[153] KINGMA D P, BA J. Adam: a method for stochastic optimization. https://arxiv.org/abs/1412.6980.
[154] DAUPHIN Y, DE VRIES H, BENGIO Y. Equilibrated adaptive learning rates for non-convex optimization. Proc. of the 28th International Conference on Neural Information Processing Systems, 2015, 1: 1504–1512.
[155] DUCHI J, HAZAN E, SINGER Y. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 2011, 12(7): 2121–2159.
[156] BYRD R H, NOCEDAL J, SCHNABEL R B. Representations of quasi-Newton matrices and their use in limited memory methods. Mathematical Programming, 1994, 63(1): 129–156.
[157] LI T, TAN L, HUANG Z H, et al. Low dimensional trajectory hypothesis is true: DNNs can be trained in tiny subspaces. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2022, 45(3): 3411−3420.
[158] SZE V, CHEN Y H, YANG T J, et al. How to evaluate deep neural network processors: TOPS/W (alone) considered harmful. IEEE Solid-State Circuits Magazine, 2020, 12(3): 28–41.
[159] HOROWITZ M. 1.1 computing's energy problem (and what we can do about it). Proc. of the IEEE International Solid-State Circuits Conference Digest of Technical Papers, 2014: 10−14.
[160] SZE V, CHEN Y H, YANG T J, et al. Efficient processing of deep neural networks: a tutorial and survey. Proceedings of the IEEE, 2017, 105(12): 2295–2329.

Biographies

OU Xinwei was born in 2000. She received her B.S. degree in electronic information engineering from Xidian University, Xi ’an, China, in 2022. She is working towards her M.S. degree with University of Electronic Science and Technology of China, Chengdu, China. Her research interests include tensors for efficient deep learning.
E-mail: xinweiou@std.uestc.edu.cn

CHEN Zhangxin was born in 1978. He received his M.S. degrees and Ph.D. degrees from University of Electronic Science and Technology of China, both in communication and information system, in 2003 and 2009, respectively. From 2012, he has been an associate professor at the Department of Electronic Engineering, University of Electronic Science and Technology of China. His research interests focus on signal processing in distributed radar system and airborne radar system.
E-mail: zhangxinchen@uestc.edu.cn

ZHU Ce was born in 1961. He received his B.S. degree in communication engineering from Sichuan University, Chengdu, China, in 1989, and M.E. and Ph.D. degrees from Southeast University, Nanjing, China, in 1992 and 1994, respectively, all in electronic and information engineering. He has been with the University of Electronic Science and Technology of China, Chengdu, China, as a professor since 2012. His research interests include video coding and communications, video analysis and processing, three-dimensional video, and visual perception and applications.
E-mail: eczhu@uestc.edu.cn

LIU Yipeng was born in 1983. He received his B.S. degree in biomedical engineering and Ph.D. degree in information and communication engineering from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2006 and 2011, respectively. Since 2014, he has been an associate professor with UESTC, Chengdu, China. His research interest is tensor for data processing.
E-mail: yipengliu@uestc.edu.cn