Computers, Materials & Continua CMC, vol.62, no.1, pp.321-336, 2020

Parameters Compressing in Deep Learning

Shiming He[1], Zhuozhou Li[1], Yangning Tang[1], Zhuofan Liao[1], Feng Li[1,*] and Se-Jung Lim[2]

Abstract: With the popularity of deep learning tools in image decomposition and natural language processing, how to support and store a large number of parameters required by deep learning algorithms has become an urgent problem to be solved. These parameters are huge and can be as many as millions. At present, a feasible direction is to use the sparse representation technique to compress the parameter matrix to achieve the purpose of reducing parameters and reducing the storage pressure. These methods include matrix decomposition and tensor decomposition. To let vector take advance of the compressing performance of matrix decomposition and tensor decomposition, we use reshaping and unfolding to let vector be the input and output of Tensor-Factorized Neural Networks. We analyze how reshaping can get the best compress ratio. According to the relationship between the shape of tensor and the number of parameters, we get a lower bound of the number of parameters. We take some data sets to verify the lower bound.

Keywords: Deep neural network, parameters compressing, matrix decomposition, tensor decomposition.

1 Introduction

Deep learning is the most useful tool for may applications, such as image recognize [Zhang, Yang, Li et al. (2018); Zhang, Jin, Sun et al. (2018); Chen, Xu, Zuo et al. (2018)], nature language processing [Zeng, Dai, Li et al. (2018); Xiang, Zhao, Li et al. (2018)]. But huge computation power and millions of parameters are needed in large models, which may cannot be supported and stored. It prevents the deep learning from be applied in mobile devices [Li, Liu, Wang et al. (2018); Li, Chen, Gao et al. (2018)]. How to compress the parameter has become an urgent problem to be solved. For solving this problem, there are four categories technology [Cheng, Wang, Zhou et al. (2017)]: low-rank factorization, parameter pruning, quantization, and knowledge distillation.

Low-rank factorization from sparse representation is the basic technology for network compressing. Some works try to compress the dense weight matrices with matrix decomposition [Pilászy, Takács, Németh et al. (2008); Xie, Wang, Wang et al. (2018); Lathauwer, Moor, Vandewalle et al. (2000)], and tensor decomposition [Kolda and Bade

[1]School of Computer and Communication Engineering, Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation, Changsha University of Science and Technology, Changsha, 410114, China.

[2] Liberal Arts & Convergence Studies, Honam University, Gwangju, 62399, Korea.

[*] Corresponding Author: Se-Jung Lim. Email: sejunglim@126.com.

(2009)]. Because of the significant redundancy of weight matrix, [Misha, Babak, Dinh et al. (2013)] exploits low-rank matrix decomposition to predict weight parameters. Then, tensor decomposition is introduced to treat the high order parameters. Several kinds of tensor decomposition are used to compress parameters, such as generalized singular value decomposition (GSVD) [Zhang, Zou, He et al. (2015); Tai, Xiao, Zhang et al. (2016)], CP (Canonical decomposition/Parallel factor) decomposition [Lebedev, Ganin, Rakhuba et al. (2014); Xie, Peng, Wang et al. (2018); Xie, Li, Wang et al. (2018)], tucker decomposition [Xie, Li, Wang et al. (2017); Kim, Park, Yoo et al. (2016)], tensor train decomposition [Tjandra, Sakti, Nakamura et al. (2017); Novikov, Podoprikhin, Osokin et al. (2015); Tjandra, Sakti, Nakamura et al. (2018)], and block term decomposition [Chen, Jin, Kang et al. (2018)].

For full connection layer, most of above works are designed to compress parameters for vector inputs or matrix inputs. In real life, there are always multi-way features from different perspectives which can be treated as tensors. To jointly perform factorization and training of an NN, chien et al. [Chien and Bao (2018)] use the tucker decomposition to replace the affline transformation in a neural network, and builds a tensor-factorized multi-layer perceptron (MLP), named tensor-factorized neural networks (TFNN). TFNN allows the tensor inputs and preserves the tensor structure from input layer to hidden layer. It can significantly reduce the number of parameters and time complexity. However, this improvement is not guaranteed for vector inputs, because of only one-way in the input.

With MLP, we usually unfold the high order tensor input into a one-way vector. With tensor-factorized MLP, can we reshape the one-way vector input into a high order tensor to get this improvement? Inspired by this, this paper explores the reshaping and unfolding to flexibly use the TFNN for further compressing parameters. The main contributions of this work can be summarized as follows.

- To take advance of the good compressing performance of matrix decomposition and tensor decomposition, we let a vector be reshaped into a matrix or tensor as the input of the TFNN, and the output of the TFNN be unfolded into a vector.
- We analyze how reshaping can get the best compress ratio. According to the relationship between the shape of tensor and the number of parameters, we can get a lower bound of the number of order and dimensions of the tensor without integer constraint and with integer constraint.
- We take some data sets to verify the lower bound.

The remainder of this paper is organized as follows. The related works is reviewed in Section 2. Section 3 introduces the preliminaries of tensor and tensor decomposition. Section 4 introduces the basic NN and tensor-factorized neural network. The detail of parameters compressing neural networks and analysis of the parameter compressed ratio are present in Section 5. Section 6 provides simulation results. In the end, we conclude this work.

2 Related works

For compressing and accelerating model of deep neural networks, some works tried to compress the dense weight matrices with sparse representations technologies. We classify these works by the technologies that these works used.

Matrix decomposition: Misha et al. [Misha, Babak, Dinh et al. (2013)] demonstrates that significant redundancy generally exists in the deep learning models. According to the redundancy, this work learns only a part of weights and predicts the rest by low-rank matrix decomposition.

Tensor decomposition: Zhang et al. [Zhang, Zou, He et al. (2015)] uses generalized singular value decomposition to solve the reconstruction method that takes into account the nonlinear neurons and a low-rank constraint. Tai et al. [Tai, Xiao, Zhang et al. (2016)] speeds up the CNN and reduces the redundancy of convolution kernels by a new algorithm of computing the low-rank tensor decomposition. Lebedev et al. [Lebedev, Ganin, Rakhuba et al. (2014)] uses a sum of a small number of rank-one tensors to approximate the 4D convolutional kernels tensor according to the low-rank CP-decomposition.

To apply deep CNNs on mobile devices, Kim et al. [Kim, Park, Yoo et al. (2016)] uses tucker decomposition on each kernel tensor and the decomposition rank is dependent on a global analytic solution of variational Bayesian matrix factorization. Chien et al. [Chien and Bao (2018)] use the tucker decomposition to replace the affline transformation in a neural network, named tensor-factorized neural networks. It preserves the tensor structure and builds a tensor-factorized MLP, for which it can allow the tensor inputs.

Novikov et al. [Novikov, Podoprikhin, Osokin et al. (2015)] use the tensor train (TT) format instead of the low rank matrix decomposition to represent the weight matrix of the fully connected layer in the CNN. Tjandra et al. [Tjandra, Sakti, Nakamura et al. (2017)] propose a TT based RNN architecture, which redefines two different RNNs using the TT format: simple RNN and GRU RNN. Tjandra et al. [Tjandra, Sakti, Nakamura et al. (2018)] study the performance of several different parameter compression methods, such as CP decomposition, tucker decomposition and TT decomposition. This work demonstrates that TT based GRU can obtain the best performance within different parameters.

Chen et al. [Chen, Jin, Kang et al. (2018)] proposes a new architecture to enhance the parameter efficiency of deep neural networks through block term decomposition, in which a high order tensor is approximated in a sum of several low-rank tuckers. It shares knowledge across different residual units by shared factors.

3 Preliminaries

For clarification, we firstly describe the notation as follows. We use lowercase letters (*a, b,...*), boldface lowercase (**a, b,**...), and boldface capitals (**A**, **B**,...) to represent scalars, vectors, and matrices, respectively. Higher-order tensors are represented as calligraphic letters ($\mathcal{X}$,$\mathcal{Y}$, ...).

3.1 Preliminaries of tensor

Definition 1 (Tensor)**.** A *tensor* is a multidimensional array, which can be considered as a higher-order generalization of a vector (first-order tensor) and a matrix (second-order tensor). For an *M*-way or *M*th-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_M}$, M is the order of $\mathcal{X}$, also known as way or mode, and I_i is the dimension of *i*th way. The element $(i_1, i_2, \cdots, i_m)$ of $\mathcal{X}$ is denoted by $x_{i_1 i_2 \cdots i_m}, i_m \in \{1, 2, \cdots I_m\}, 1 \le m \le M$.

Definition 2 (Slice and Mode-*m* unfolding**).** When fixing all indexes except two indexes, the tensors can be divided into slices which are matrices. For a three-way tensor $\mathcal{X}$, there are three kinds of slices, such as horizontal slices, lateral slices and frontal slices, which are denoted by $\mathbf{X}_{i::}$, $\mathbf{X}_{:j:}$, and $\mathbf{X}_{::k}$, respectively.

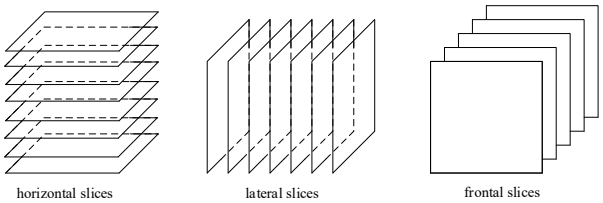


Figure 1: Slices of three-way tensor

The Kiers unfolding or matricizing of tensor is the mosaics of slices, as shown in Fig. 2. The mode-*m* unfolding of $\mathcal{X}$ is denoted by $\mathbf{X}_{(m)}$.

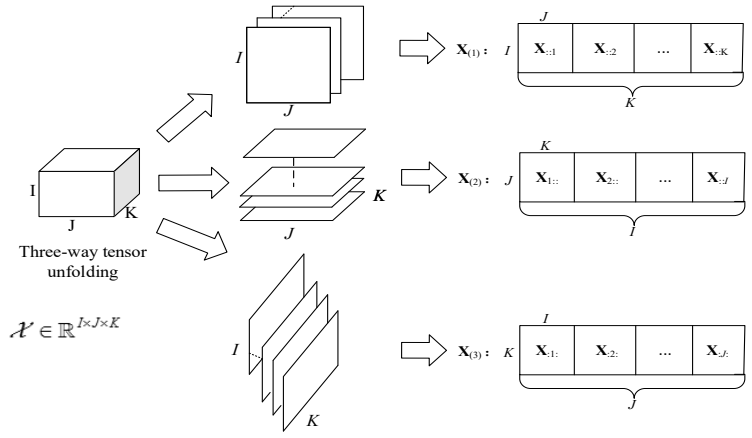


Figure 2: Unfolding of three-way tensor

Definition 3 (Outer product of vectors**).** For *M* vectors $\boldsymbol{x}^{(i)} \in \mathbb{R}^{i\times 1}, i = 1,\cdots,m$, the outer product of these vectors is a *M*-way tensor, denoted as $\boldsymbol{x}^{(1)} \circ \boldsymbol{x}^{(2)} \circ \cdots \circ \boldsymbol{x}^{(m)}$. Or It can be defined in element form as $(\boldsymbol{x}^{(1)} \circ \boldsymbol{x}^{(2)} \circ \cdots \circ \boldsymbol{x}^{(m)})_{i_1 i_2 \cdots i_m} = x_{i_1}^{(1)} x_{i_2}^{(2)} \cdots x_{i_m}^{(m)}$, where $x_j^{(i)}$ is the *j*th element of vector $\boldsymbol{x}^{(i)}$.

Definition 4 (Rank one tensor**).** A 3-way tensor $\mathcal{X}$ is a rank one tensor if it can be written as the outer product of three vectors, i.e. $\mathcal{X} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c}$.

Definition 5 (*m*-mode product**).** The *m*-mode product of a tensor $\mathcal{X} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_M}$ and a matrix $\mathbf{U}^{(m)} \in \mathbb{R}^{J_m\times I_m}$ is denoted by $\mathcal{X} \times_m \mathbf{U}^{(m)}$, which is a $I_1 \times \cdots \times I_{m-1} \times J_m \times I_{m+1} \times \cdots \times I_M$ tensor and the element is defined as

$$\left(\mathcal{X} \times_m \mathbf{U}^{(m)}\right)_{i_1\cdots i_{m-1} j i_{m+1}\cdots i_M} = \sum_{i_m=1}^{I_m} x_{i_1 i_2 \cdots i_M} u_{j i_m} . \tag{1}$$

For example, $\mathcal{X}$ is a 3-way tensor as shown in Fig. 3. U is a matrix and the value is set to $\mathbf{U}=\begin{bmatrix}1 & 3 & 5 & 7\\ 2 & 4 & 6 & 8\end{bmatrix}$. The 2-mode product $\mathcal{B}=\mathcal{X}\times_2 \mathrm{U}\in\mathbb{R}^{3\times 2\times 2}$ is a tensor, whose mode-2 unfolding matrix is

$$\mathbf{B}_{(2)}=\left[\begin{array}{ccc|ccc}236 & 268 & 300 & 620 & 652 & 684\\ 280 & 320 & 360 & 760 & 800 & 840\end{array}\right]. \tag{2}$$

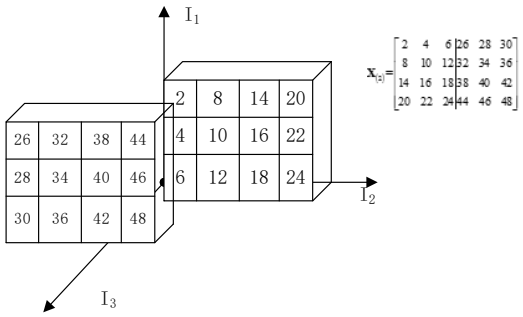


Figure 3: A tensor $\mathcal{X}\in\mathbb{R}^{3\times 4\times 2}$

In addition, the following properties of the *k*-mode matrix product are true.

$$\mathcal{X}\times_m \mathrm{X}\times_n \mathrm{Y}=\mathcal{X}\times_n \mathrm{Y}\times_m \mathrm{X}\quad(m\neq n) \tag{3}$$

$$\mathcal{X}\times_m \mathrm{X}\times_m \mathrm{Y}=\mathcal{X}\times_m(\mathrm{YX}) \tag{4}$$

3.2 Tensor decomposition

Tensor decomposition is a method used to analyze multi-channel data structures. We introduce two basic tensor decomposition models, one is CP model, the other is tucker model. The CP model can be considered as a special implementation of tucker model, in which the core tensor is hyper diagonal.

3.2.1 CP-decomposition

CP-decomposition is also called as CANDECOMP/PARAFAC model, as shown in Fig. 4. Inspired of matrix fraction ($x_{ij}=\sum_{r=1}^{R}a_{ir}b_{ir}$), an element of a three-way tensor $\mathcal{X}\in\mathbb{R}^{I_1\times I_2\times I_3}$ can be calculated by the sum of a finite number of multiply of three scalars,

$$x_{ijk}=\sum_{r=1}^{R}a_{ir}b_{jr}c_{kr} \tag{5}$$

where $R>0$. Thus, the three-way tensor can be expressed as the sum of R rank one tensors which can be calculated by the outer product of three vectors.

$$\mathcal{X}=\sum_{r=1}^{R}\boldsymbol{a}_r\circ\boldsymbol{b}_r\circ\boldsymbol{c}_r \tag{6}$$

where $\boldsymbol{a}_r = [a_{1r},...,a_{I_1 r}]^T \in \mathbb{R}^{I_1 \times 1}, \boldsymbol{b}_r = [b_{1r},...,b_{I_2 r}]^T \in \mathbb{R}^{I_2 \times 1}, \boldsymbol{c}_r = [c_{1r},...,c_{I_3 r}]^T \in \mathbb{R}^{I_3 \times 1}$ are the factor vectors. Further, we can get the tenor's factor matrices.

$$\mathbf{A} = [\boldsymbol{a}_1,...,\boldsymbol{a}_R] \in \mathbb{R}^{I_1 \times R}, \mathbf{B} = [\boldsymbol{b}_1,...,\boldsymbol{b}_R] \in \mathbb{R}^{I_2 \times R}, \mathbf{C} = [\boldsymbol{c}_1,...,\boldsymbol{c}_R] \in \mathbb{R}^{I_3 \times R} \tag{7}$$

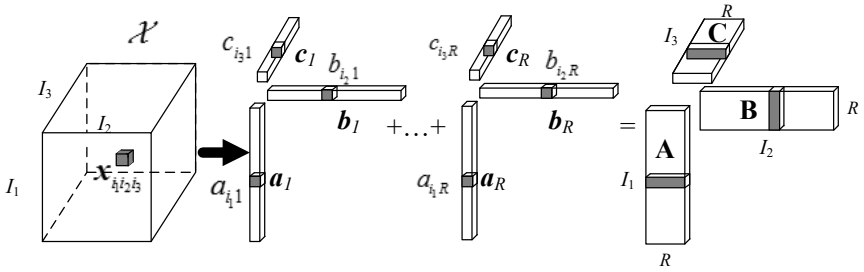


Figure 4: The CP-decomposition of three-way tensor

3.2.2 Tucker tensor decomposition

For a three-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, the tucker tensor decomposition is to divide the tensor into the produce of a core-tensor and three factor matrices.

$$\mathcal{X} = \mathcal{A} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \mathbf{U}^{(3)} \tag{8}$$

where $\mathcal{A} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$, $\mathbf{U}^{(1)} \in \mathbb{R}^{J_1 \times I_1}, \mathbf{U}^{(2)} \in \mathbb{R}^{J_2 \times I_2}, \mathbf{U}^{(3)} \in \mathbb{R}^{J_3 \times I_3}$ are the core-tensor and three factor matrices, $\times_k$ is the mode-*k* produce, as shown in Fig. 5. Of course, its elements are expressed as follow:

$$x_{i_1 i_2 i_3} = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \sum_{j_3=1}^{J_3} a_{j_1 j_2 j_3} u_{i_1 j_1}^{(1)} u_{i_2 j_2}^{(2)} u_{i_3 j_3}^{(3)} \tag{9}$$

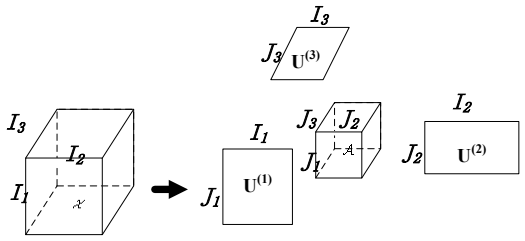


Figure 5: The tucker tensor decomposition of three-way tensor

Be universally known, the tucker decomposition in tensors is a multi-linear extension of the SVD. There is a conversion relation between High-order SVD and the tucker decomposition.

4 Unfolding based deep neural network and tensor-factorized neural network

In this section, we introduce the basic NN and tensor-factorized neural network.

4.1 Unfolding based deep neural network

Fig. 6 is the structure of full connection layer with the transformations of $\boldsymbol{h}=g(\boldsymbol{v}\mathbf{W})$,

where $\boldsymbol{v}$ is an input vector with I_1-dimensional input, $\boldsymbol{h}$ is an output vector with J_1-dimensional, and $\mathbf{W}$ is an $I_1 \times J_1$ matrix of parameters. And the number of parameters on this layer is $I_1 J_1$.

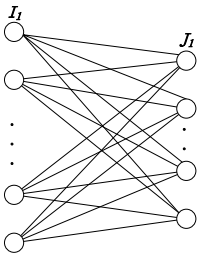


Figure 6: The full connection layer of NN

For matrix or tensor input, the matrix or tensor needs to be unfolded into vector and then fed into the full connection layer. For a $I_1 \times I_2$ matrix input and a $J_1 \times J_2$ matrix output, it is unfolded into an input vector with $I_1 I_2$-dimensional and an output vector with $J_1 J_2$-dimensional. The weight matrix is an $I_1 I_2 \times J_1 J_2$ matrix. And the number of parameters on this layer is $I_1 I_2 J_1 J_2$. Similarly, for the three-way and N-way tensor input and output, the number of parameters are $I_1 I_2 I_3 J_1 J_2 J_3$ and $\prod_{n=1}^{N} I_n J_n$, respectively.

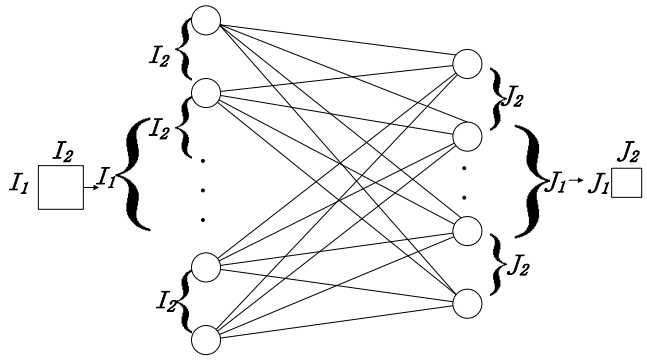


Figure 7: The full connection layer on matrix input and output by unfolding

4.2 Tensor-factorized neural network

According to the tucker decomposition introduced in Section 3.2.2, a N-way input tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \ldots \times I_N}$ is converted to a core tensor $\mathcal{A} \in \mathbb{R}^{J_1 \times J_2 \times \ldots \times J_N}$ with N factor matrices. Based on the Eq. (9), we can consider the core tensor $\mathcal{A}$ as a tensor weight. However, the decomposition solution is non-unique. We may need to find a unique solution with some constraints. From the other point of view, we can get the inverse of the tucker decomposition by the Eq. (10), as shown in Fig. 8.

$$\mathcal{X} = \mathcal{A} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times_3 \cdots \times_N \mathbf{U}^{(N)} \Rightarrow \mathcal{A} = \mathcal{X} \times_1 \mathbf{U}^{(1)+} \times_2 \mathbf{U}^{(2)+} \times_3 \cdots \times_N \mathbf{U}^{(N)+} \quad (10)$$

In the above equation, $\mathbf{U}^{(n)+}=(\mathbf{U}^{(n)\mathrm{T}}\mathbf{U}^{(n)})^{-1}\mathbf{U}^{(n)\mathrm{T}}$ is the pseudo inverse of $\mathbf{U}^{(n)}$. When the dimensions $J_1,J_2,...,J_N$ of the core tensor are less than the dimensions $I_1,I_2,...,I_N$ of the original tensor, we can treat the core tensor $\mathcal{A}$ as a compressed version of the tensor $\mathcal{X}$. Thus, the core tensor $\mathcal{A}$ extracts the features of the tensor $\mathcal{X}$ via N factor matrices $\{\mathbf{U}^{(1)+},\cdots,\mathbf{U}^{(N)+}\}$.

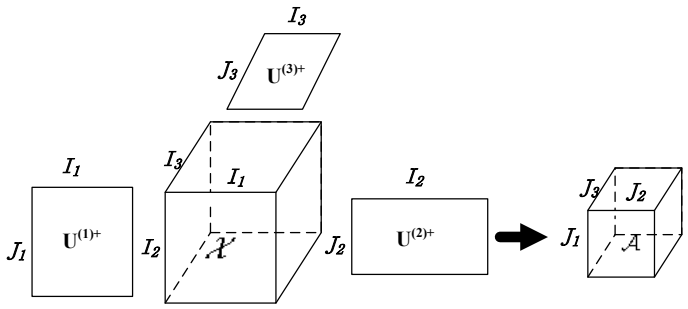


Figure 8: The inverse of the three-way tucker decomposition

According to the inverse of tucker decomposition, Chien et al. [Chien and Bao (2018)] propose a tensor-factorized MLP named Tensor-Factorized Neural Networks (TFNN), which can take the tensor as input and output of full connected layer directly, as shown in Fig. 9. The original tensor $\mathcal{X}$ is the input tensor and the core tensor $\mathcal{A}$ is the output tensor. There are $J_1J_2J_3$ weight tensors that have the same dimensions and way with the original input tensor. The inner product of the input tensor and a weight tensor is the value of one element on the output tensor. Specially, all the weight tensors are the rank-one tensor and they share the vectors which are the row of the pseudo inverse matrices $\mathbf{U}^{(1)+}$, $\mathbf{U}^{(2)+}$, $\mathbf{U}^{(3)+}$ from the tucker decomposition.

TFNN tightly combines TF and NN. For the three-way tensor, the three-way TFNN is constructed by decomposing and activating the input tensor $\mathcal{X}$ into a feature tensor $\mathcal{A}$ using three factor matrices $\mathbf{U}^{(1)+}$, $\mathbf{U}^{(2)+}$, $\mathbf{U}^{(3)+}$. Chien et al. [Chien and Bao (2018)] also gives the back propagation procedure of TFNN. Similar to the 3-way decomposition, the N-way tensor input can be decomposed to construct an N-way TFNN by N factor matrices $\mathbf{U}^{(1)+},\cdots,\mathbf{U}^{(N)+}$.

TFNN use the decomposing to retain the tensor structure, while NN unfolds the N-way tensor $\mathcal{X}\in\mathbb{R}^{I_1\times I_2\times\ldots\times I_N}$ into a one-way vectors $\mathbf{x}_t\in\mathbb{R}^{I_1I_2\ldots I_N}$. As a result, all the parameters are just stored in the factor matrices. The number of parameters of two-way and three-way TFNN are $I_1J_1+I_2J_2$, $I_1J_1+I_2J_2+I_3J_3$, respectively. For the N-way tensor, the number of parameters is $\sum_{n=1}^{N}I_nJ_n$, which is significantly compressed comparing with unfolding NN in Section 4.1. It is an efficient parameter compressing method. The compressed ratio is $\prod_{n=1}^{N}I_nJ_n/\sum_{n=1}^{N}I_nJ_n$.

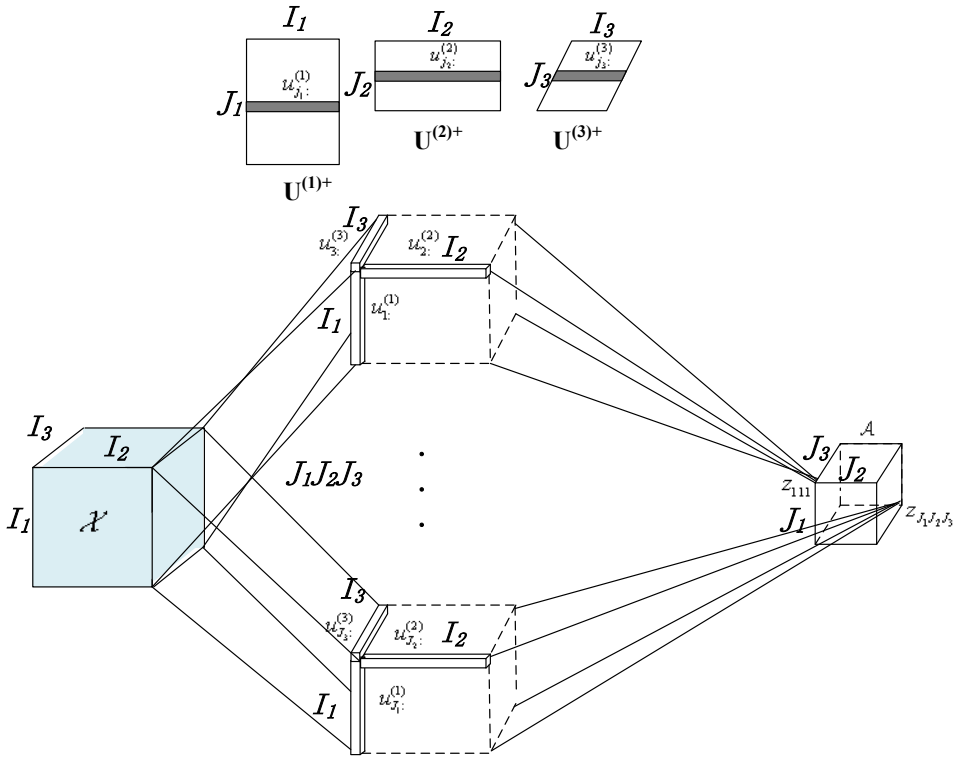


Figure 9: The three-way tensor layer with tucker decomposition

5 Parameters compressing neural networks

Based on the compressed ratio of the TFNN, we can get that when the way of tensor is large, the compressed ration is big with the same size of the input data. If a vector is reshaped into a matrix or tensor as the input of the TFNN and the output of the TFNN is unfolded into a vector, it may further control the size of parameters and improve the efficiency.

5.1 Basic ideas

As shown in Fig. 10, the input vector with $I_1^{'}$-dimension is reshaped into a three-way tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, and the output tensor $\mathcal{A} \in \mathbb{R}^{J_1 \times J_2 \times J_3}$ of the TFNN is unfolded into a output vector with $J_1^{'}$-dimension. Obviously, the number of parameters with vector input and output can be further reduced by tensor decomposition.

However, the input vector can be reshaped into two-way, three-way, and even N-way tensor. Reshaping into a two-way, three-way or N-way tensor will produce various parameter compressed ratio and representation power. After reshaping and unfolding, the representation power is dependent on the structure of the input data. Therefore, in this paper, we only focus on that how to reshaping and unfolding can get the best parameter compressed ratio.

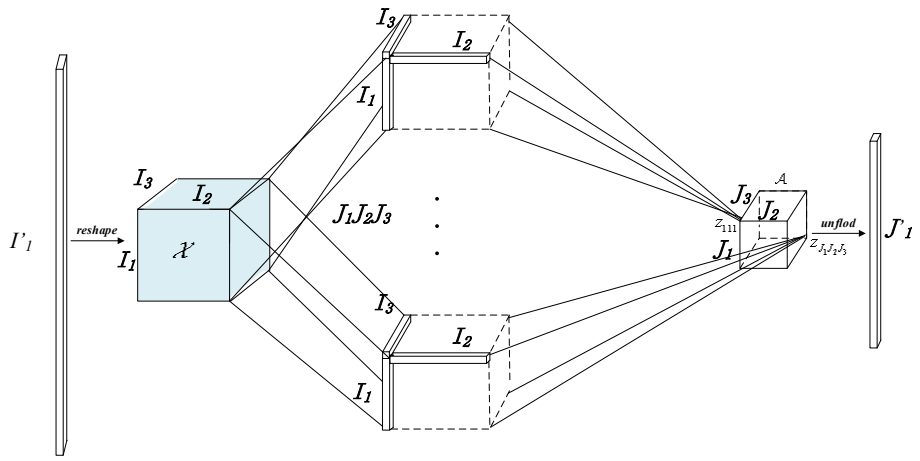


Figure 10: The parameters compressing neural network with vector input and output

Assume that the dimension of the input and output vectors are $I_1^{'}$ and $J_1^{'}$, respectively, the input vectors can be reshaped into a *N*-way tensor with $(I_1, I_2, ..., I_N)$ dimension and the *N*-way output tensor of TFNN is $(J_1, J_2, ..., J_N)$ dimension. To store the all element in the vector, the dimensions between vectors and tensors should satisfy the following constrains.

$$\prod_{n=1}^{N} I_n = I_1^{'}, \quad \prod_{n=1}^{N} J_n = J_1^{'} \tag{11}$$

The compressed ratio can be represented as $I_1^{'} J_1^{'} / \sum_{n=1}^{N} I_n J_n$.

5.2 Analysis of the parameter compressed ratio

To get the best parameter compressed ratio, we need to minimize the number of parameters in the *N*-way TFNN. The problem can be formulated as following.

$$\begin{aligned} &\min_{I,J,N} \sum_{n=1}^{N} I_n J_n \\ &s.t. \prod_{n=1}^{N} I_n \geq I_1^{'} \\ &\quad \prod_{n=1}^{N} J_n \geq J_1^{'} \\ &\quad I_n, J_n \in \mathbb{Z}, \text{ for } n \\ &\quad N \in \mathbb{Z} \end{aligned} \tag{12}$$

where I_n and J_n is the *n*-th dimensional of tensors, and *N* is the way of the input and output tensors of TFNN.

5.2.1 Without integer constraint

In this sub-section, we ignore the integer constrain of I_n, J_n . Specially, if all the dimension J_n of output tensor are equal, the number of weight parameters is dependent on the sum of the links' length of input tensor. Intuitively, the sum of the links' length of the super-cube is minimum, that is, $I_n = \left(I_1^{'}\right)^{1/N}$.

If N is given and we ignore the integer constrain, the best solution of problem (12) is $N\left(I_1^{'}J_1^{'}\right)^{1/N}$, when the dimension of each way satisfies the following equation.

$$I_n J_n = \left(I_1^{'}J_1^{'}\right)^{1/N}, \text{ for } n \tag{13}$$

Proof: When N is given, the problem of (12) without the integer constrain is transformed into problem (14).

$$\begin{aligned} &\min_{I,J} \sum_{n=1}^{N} I_n J_n \\ &s.t. \prod_{n=1}^{N} I_n = I_1^{'} \\ &\quad \prod_{n=1}^{N} J_n = J_1^{'} \end{aligned} \tag{14}$$

In order to solve it, we get the Lagrange function of the problem (14) as follows.

$$L = \sum_{n=1}^{N} I_n J_n + \alpha(\prod_{n=1}^{N} I_n - I_1^{'}) + \beta(\prod_{n=1}^{N} J_n - J_1^{'}) \tag{15}$$

where α, β are the Lagrange multipliers. And then we can solve it by KKT (Karush Kuhn Tucker) conditions.

$$\begin{aligned} &\partial L / \partial I_n = J_n + \alpha \prod_{i=1, i\neq n}^{N} I_i = 0, \quad \text{for } n \\ &\partial L / \partial J_n = I_n + \beta \prod_{i=1, i\neq n}^{N} J_i = 0, \quad \text{for } n \\ &\alpha \neq 0, \beta \neq 0 \\ &\prod_{n=1}^{N} I_n - I_1^{'} = 0, \prod_{n=1}^{N} J_n - J_1^{'} = 0 \end{aligned} \tag{16}$$

The optimal dimension relationships between the reshaped input tensor and output tensor are as follows.

$$I_1 J_1 = I_2 J_2 = \cdots = I_n J_n, \text{ for } n \tag{17}$$

At the same time, to satisfy the storing constrains ($\prod_{n=1}^{N} I_n - I_1^{'} = 0, \prod_{n=1}^{N} J_n - J_1^{'} = 0$), we can get that the multiple of $I_n J_n$ is $\left(I_1^{'}J_1^{'}\right)^{1/N}$.

Then, we can get the best way of tensor N^*.

$$\min_{I,J,N}\sum_{n=1}^{N} I_n J_n = \min_{N}\sum_{n=1}^{N}\left(I_1^{'}J_1^{'}\right)^{\frac{1}{N}} = \min_{N} N\left(I_1^{'}J_1^{'}\right)^{\frac{1}{N}}$$

$$\partial N\left(I_1^{'}J_1^{'}\right)^{\frac{1}{N}} \Big/ \partial N = \left(I_1^{'}J_1^{'}\right)^{\frac{1}{N}}\left(1-\ln\left(I_1^{'}J_1^{'}\right)/N\right) = 0 \tag{18}$$

$$N^* = \ln\left(I_1^{'}J_1^{'}\right)$$

The number of parameters is

$$\sum_{n=1}^{N} I_n J_n = \sum_{i=1}^{N}\left(I_1^{'}J_1^{'}\right)^{1/N} = N\left(I_1^{'}J_1^{'}\right)^{1/N} = \ln\left(I_1^{'}J_1^{'}\right)\left(I_1^{'}J_1^{'}\right)^{1/\ln\left(I_1^{'}J_1^{'}\right)}. \tag{19}$$

5.2.2 With integer constraint

Considering integer constraints, we need to do a rounding operation to N^*.

$$N^* = \left\lceil \ln\left(I_1^{'}J_1^{'}\right)\right\rceil \text{ or } N^* = \left\lfloor \ln\left(I_1^{'}J_1^{'}\right)\right\rfloor \tag{20}$$

where $\lceil\bullet\rceil, \lfloor\bullet\rfloor$ are the ceiling and floor functions. The optimal N^* may be one of the two candidate values. Therefore, we take the two values as N^* to calculate the result separately. In each value of N^*, we set the default value of I_n, J_n as the ceiling of $\left(I_1^{'}\right)^{1/N^*}$, $\left(J_1^{'}\right)^{1/N^*}$. And then, with the constrain of storing all input and out data in Eq. (11), we orderly reduce the value of I_n, J_n to the floor of $\left(I_1^{'}\right)^{1/N^*}$, $\left(J_1^{'}\right)^{1/N^*}$. After reducing, we check whether switching the value of J_n with different subscript will further reduce the object value. If yes, we switch the value of J_n. Finally, we compare the the object value of the two candidate values to get the optimal number of parameters. The overall algorithm is described as Algorithm 1.

Algorithm 1 Reduced Iteration Algorithm

Input: The dimensions of the input and output vectors $I_1^{'}$, $J_1^{'}$

Output: The optimal number of parameters

1: $N^* = \left\lceil \ln\left(I_1^{'}J_1^{'}\right)\right\rceil, I_n = \left\lceil \left(I_1^{'}\right)^{1/N^*}\right\rceil$, $J_n = \left\lceil \left(J_1^{'}\right)^{1/N^*}\right\rceil$

2: for $n = 1$ to N^* do

3: if $I_1 \times I_2 \times \cdots (I_n - 1) \times \cdots I_{N^*} \le I_1^{'}$ then break

4: $I_n - - = 1$

5: for $n = 1$ to N^* do

6: if $J_1 \times J_2 \times \cdots (J_n - 1) \times \cdots J_{N^*} \le J_1^{'}$ then break

7: $J_n -= 1$

8: end for

9: for $a = 1$ to N^* do

10: for $b = a+1$ to N^* do

11: if $I_a J_a + I_b J_b > I_a J_b + I_b J_a$ then

12: swap J_a, J_b

13: end if

14: end for

15: end for

16: $S_1 = \sum_{n=1}^{N^*} I_n J_n$

17: $N^* = \left\lfloor \ln\left(I_1' J_1'\right) \right\rfloor, I_n = \left\lceil \left(I_1'\right)^{1/N^*} \right\rceil, \; J_n = \left\lceil \left(J_1'\right)^{1/N^*} \right\rceil$

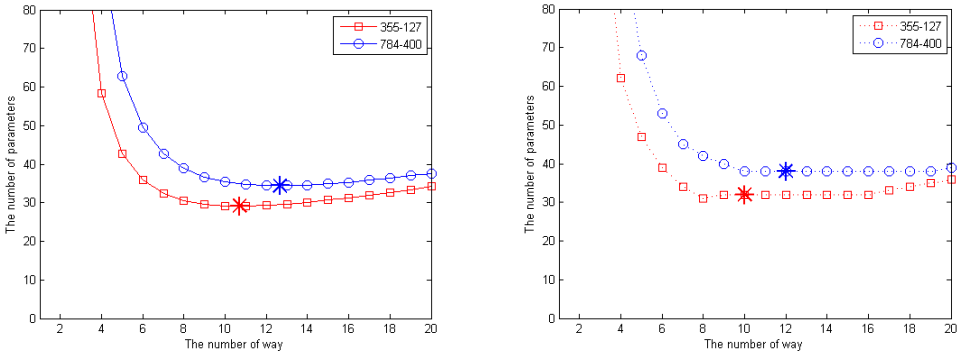
18: Repeat steps 2 to 15

19: $S_2 = \sum_{n=1}^{N^*} I_n J_n$

20: return $\min\left(S_1, S_2\right)$

6 Simulation

We use the input and output sizes of different data sets for evaluation in Tab. 1. First is from Qian et al. [Qian, Fan, Hu et al. (2014)] where the input vector contains 355 dimensions and the output vector contains 127 dimensions. Second is the Mixed National Institute of Standards and Technology (MNIST) database [Lecun, Bottou, Bengio et al. (1998)] which consists of gray-scale images of size 28×28 and a layer of the output matrix is 20×20. We only consider the vector size of MNIST. The metric is the number of parameters after reshaping. Fig. 11 shows the number of parameters according to Eq. (14) with reshaping to N-way tensor from 1 to 20. Fig. 11(a) is the number of parameters without integer constraint, (b) is the number of parameters with integer constrain. We can see that there are two stages, the quick decreasing and the slow increasing. Tab. 1 shows the reshaping result of data sets obtained by the Eq. (19) and Algorithm 1, in which the best value of N is same as the Fig. 11.



(a) (b)

Figure 11: The number of parameters with different way of tensor: (a) without integer constraint, (b) with integer constraint

Table 1: The reshaping result of data sets with different input and output size

	Data set	[Qian, Fan, Hu et al. (2014)]	MNIST [Lecun, Bottou, Bengio et al. (1998)]
	Input size	355	784(28×28)
	Output size	127	400(20×20)
N^* /object value	I_n J_n N without integer constraint	10.72/29.13	12.66/34.4
	only N with integer constraint	11/29.13	13/34.41
	I_n J_n N with integer constraint	10/32	12/38
	I_n	1,2,2,2,2,2,2,2,2,2,	1,1,2,2,2,2,2,2,2,2,2,2
	J_n	2,2,2,2,2,2,2,1,1,1	2,2,2,2,2,2,2,2,2,1,1,1

From the Fig. 11(b), our algorithm cannot always get the optimal result. The optimal N for Qian et al. [Qian, Fan, Hu et al. (2014)] is 8 and object value is 31. It is noticed that it can lead to a N^*-way tensor. This shape may not be suitable for all data sets. But this is a lower bound. To trade off the representation power and the number of parameters, we can consider the way of tensor from 2 to N^*.

7 Conclusion

In order to control the size of parameters and improve the training efficiency, we let a vector be reshaped into a matrix or tensor as the input of the TFNN. We analyse how reshaping can get the best compress ratio. According to the relationship between the shape of tensor and the number of parameters, we get a lower bound of the number of parameters. We take some data sets to verify the lower bound. The future work is to

choose a better shape of tensor for keeping representation power.

Acknowledgement: This work was supported by National Natural Science Foundation of China (Nos. 61802030, 61572184), the Science and Technology Projects of Hunan Province (No. 2016JC2075), the International Cooperative Project for "Double First-Class", CSUST (No. 2018IC24).

References

Chien, J. T.; Bao, Y. T. (2018): Tensor-factorized neural networks. *IEEE Transactions on Neural Networks & Learning Systems*, vol. 29, no. 5, pp. 1998-2011.

Chen, Y. P.; Jin, X. J.; Kang, B. Y.; Feng, J. S.; Yan, S. C. (2018): Sharing residual units through collective tensor factorization in deep neural networks. *International Joint Conference on Artificial Intelligence*, pp. 635-641.

Chen, Y. T.; Xu, W. H.; Zuo, J. W.; Yang, K. (2018): The fire recognition algorithm using dynamic feature fusion and IV-SVM classifier. *Cluster Computing*, pp. 1-11.

Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. (2017): A survey of model compression and acceleration for deep neural networks. *IEEE Signal Processing Magazine*.

Kolda, T. G.; Bader, B. W. (2009): Tensor decomposition and applications. *SIAM Review*, vol. 51, no. 3, pp. 455-500.

Kim, Y. D.; Park, E.; Yoo, S.; Choi, T.; Yang, L. (2016): Compression of deep convolutional neural networks for fast and low power mobile applications. *Computer Vision and Pattern Recognition*.

Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998): Gradient-based learning applied to document recognition. P*roceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324.

Li, W. J.; Chen, Z. Y.; Gao, X. Y.; Liu, W.; Wang, J. (2018): Multi-model framework for indoor localization under mobile edge computing environment. *IEEE Internet of Things Journal*, pp. 1.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; Lempitsky, V. (2014): Speeding up convolutional neural networks using fine-tuned CP-decomposition. *Computer Vision and Pattern Recognition*.

Lathauwer, L. D.; Moor, B. D.; Vandewalle, J. (2000): A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253-1278.

Li, W. J.; Liu, H. Y.; Wang, L. Y.; Xiang, L. Y.; Yang, Y. J. (2018): An improved linear kernel for complementary maximal strip recovery: simpler and smaller. *Theoretical Computer Science*.

Misha, D.; Babak, S.; Dinh, L.; Ranzato, M.; Freitas, N. D. (2013): Predicting parameters in deep learning. *Advances in Neural Information Processing Systems*, pp. 2148-2156.

Novikov, A.; Podoprikhin, D.; Osokin, A.; Vetrov, D. P. (2015): Tensorizing neural networks. *Advances in Neural Information Processing Systems*, pp. 442-450.

Pilászy, I.; Takács, G.; Németh, B.; Tikk, D. (2008): Investigation of various matrix factorization methods for large recommender systems. *IEEE International Conference on Data Mining Workshops*, pp. 553-562.

Qian, Y.; Fan, Y.; Hu, W.; Soong, F. K. (2014): On the training aspects of deep neural network (DNN) for parametric TTS synthesis. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3829-3833.

Tjandra, A.; Sakti, S.; Nakamura, S. (2017): Compressing recurrent neural network with tensor train. *International Joint Conference on in Neural Networks*, pp. 4451-4458.

Tjandra, A.; Sakti, S.; Nakamura, S. (2018): Tensor decomposition for compressing recurrent neural network. *International Joint Conference on in Neural Networks*, pp. 1-8.

Tai, C.; Xiao, T.; Zhang, Y.; Wang, X. G. (2016): Convolutional neural networks with low-rank regularization. *Machine Learning*.

Xiang, L. Y.; Zhao, G. H.; Li, Q.; Hao, W.; Li, F. (2018): TUMK-ELM: a fast unsupervised heterogeneous data learning approach. *IEEE Access*, vol. 6, pp. 35305-35315.

Xie, K.; Li, X. C.; Wang, X.; Cao, J. N.; Xie, G. G. et al. (2018): On-line anomaly detection with high accuracy. *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1222-1235.

Xie, K.; Li, X. C.; Wang, X.; Xie, G. G.; Wen, J. G. et al. (2017): Fast tensor factorization for accurate internet anomaly detection. *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3794-3807.

Xie, K.; Peng, C.; Wang, X.; Xie, G. G.; Wen, J. G. et al. (2018): Accurate recovery of internet traffic data under variable rate measurements. *IEEE/ACM Transactions on Networking*, vol. 26, no. 3, pp. 1137-1150.

Xie, K.; Wang, L. L.; Wang, X.; Xie, G. G.; Wen, J. G. (2018): Low cost and high accuracy data gathering in WSNs with matrix completion. *IEEE Transactions on Mobile Computing*, vol. 17, no. 7, pp. 1595-1608.

Zeng, D. J.; Dai, Y.; Li, F.; Sherratt, R. S.; Wang, J. (2018): Adversarial learning for distant supervised relation extraction. *Computers, Materials & Continua*, vol. 55, no. 1, pp. 121-136.

Zhang, J. M.; Jin, X. K.; Sun, J.; Wang, J.; Saigaiah, A. K. (2018): Spatial and semantic convolutional features for robust visual object tracking. *Multimedia Tools and Applications*, pp. 1-21.

Zhang, D. Y.; Yang, G. B.; Li, F.; Wang, J.; Saigaiah, A. K. (2018): Detecting seam carved images using uniform local binary patterns. *Multimedia Tools and Applications*, pp. 1-16.

Zhang, X. Y.; Zou, J. H.; He, K. M.; Sun, J. (2015): Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943-1955.