IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 25, 2019, accepted October 19, 2019, date of publication October 28, 2019,
date of current version November 19, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2949814

A Survey on Tensor Techniques and Applications in Machine Learning

YUWANG JI, QIANG WANG, XUAN LI, AND JIE LIU
Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Qiang Wang (wangq@bupt.edu.cn)

This work was supported in part by the Beijing Natural Science Foundation under Grant L182037, in part by the National Natural Science Foundation of China under Grant 61871045 and Grant 61325006, in part by the Beijing Natural Science Foundation under Grant L172033, in part by the National Natural Science Foundation of China under Grant 6197106661325006, and in part by the 111 Project of China under Grant B16006.

ABSTRACT This survey gives a comprehensive overview of tensor techniques and applications in machine learning. Tensor represents higher order statistics. Nowadays, many applications based on machine learning algorithms require a large amount of structured high-dimensional input data. As the set of data increases, the complexity of these algorithms increases exponentially with the increase of vector size. Some scientists found that using tensors instead of the original input vectors can effectively solve these high-dimensional problems. This survey introduces the basic knowledge of tensor, including tensor operations, tensor decomposition, some tensor-based algorithms, and some applications of tensor in machine learning and deep learning for those who are interested in learning tensors. The tensor decomposition is highlighted because it can effectively extract structural features of data and many algorithms and applications are based on tensor decomposition. The organizational framework of this paper is as follows. In part one, we introduce some tensor basic operations, including tensor decomposition. In part two, applications of tensor in machine learning and deep learning, including regression, supervised classification, data preprocessing, and unsupervised classification based on low rank tensor approximation algorithms are introduced detailly. Finally, we briefly discuss urgent challenges, opportunities and prospects for tensor.

INDEX TERMS Machine learning, tensor decomposition, higher order statistics, data preprocessing, classification.

I. INTRODUCTION

"Tensor" was first introduced by William Ron Hamilton in 1846 and later became known to scientists through the publication of Levi-Civita's book The Absolute Differential Calculus [72]. Because of its structured representation of data format and ability to reduce the complexity of multidimensional arrays, tensor has been gradually applied in various fields, such as Dictionary Learning (Ghassemi *et al.*) [88], Magnetic Resonance Imaging(MRI) (Xu *et al.*) [148], Spectral data classification (Makantasis *et al.*) [69], and Image deblurring (Geng *et al.*) [75].

When traditional vector value data is extended to tensor value data, traditional vector value based algorithms will no longer work. Thereupon, some scientists extend the traditional vector-based machine learning algorithms to tensors, such as Support tensor machine(STM) (Tao *et al.* [27]; Biswas and Milanfar [121]; Hao *et al.* [164]), tensor fisher discriminant analysis (Lechuga) [38], tensor regression (Hoa *et al.*) [89], tensor completion (Du *et al.*) [150], and so on. Recently, a series of new algorithms based on tensor have been widely used in biomedicine and image processing. Compared with traditional vector-based algorithms, tensor-based algorithms can achieve lower computational complexity and better accuracy. Through these tensor-based algorithms, high-dimensional problems can be solved effectively, and accuracy can be improved without destroying the data structure.

The key references for this survey are (Cichocki *et al.*) [3] and (Kolda and Bader) [127]. The main purpose of this survey is to introduce basic machine learning applications related to tensor decomposition and tensor network model. Similar to matrix decomposition, tensor decomposition is used to decompose complex high-dimensional tensor into the form

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

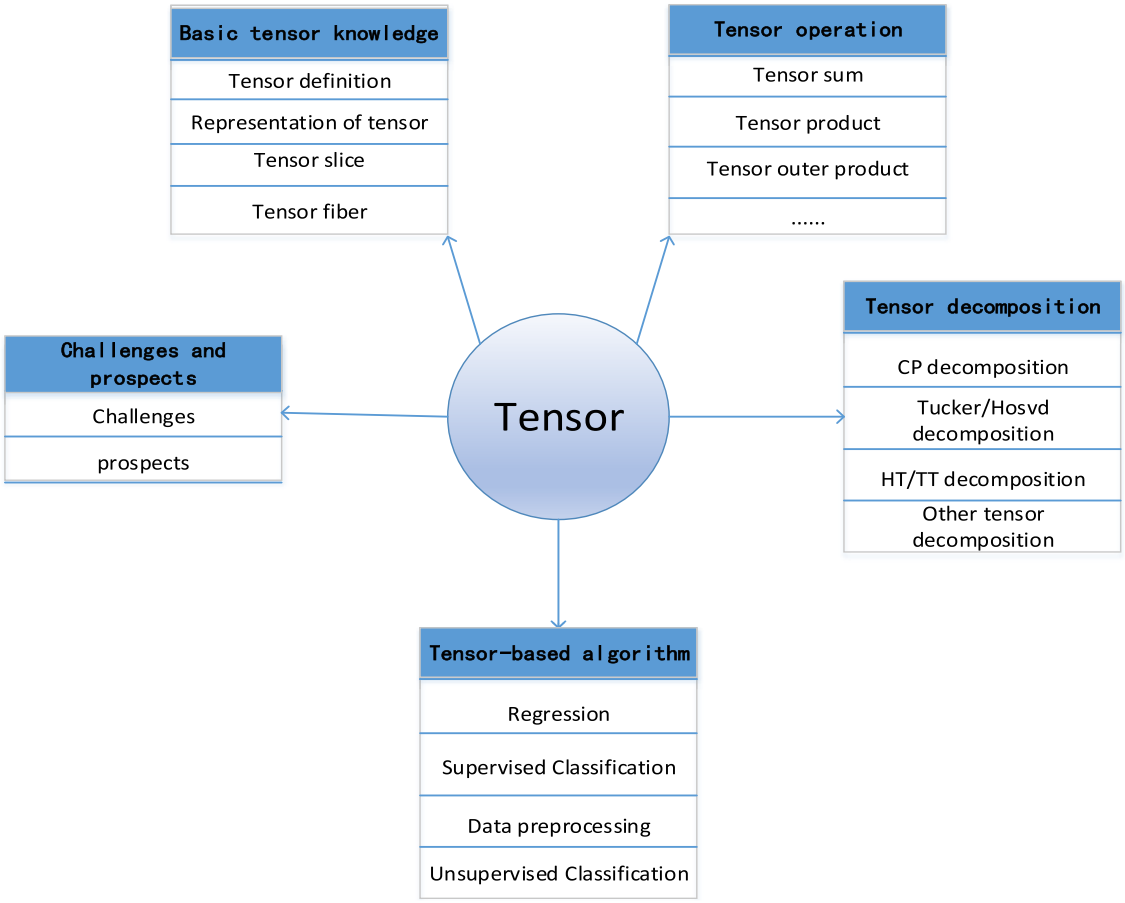

FIGURE 1. A general block diagram of the survey.

of the sum of products of factor tensor or factor vector. Tensor network decomposes the high-dimensional tensor into sparse factor matrices and low-order core tensor, which we call factors or blocks. In this way, we set the compression (that is, distributed) representation of large-size data, enhancing the advantage of interpretation and calculation.

Tensor decomposition is regarded as a sub-tensor network in this survey. That is to say, the decomposition of tensor can be used in the same way as the tensor network. We can divide the data into related and irrelevant parts by using tensor decomposition. High-dimensional big data can be compressed several times without breaking data correlation by using tensor decomposition (tensor network). Moreover, tensor decomposition can be used to reduce unknown parameters, and then the exact solution can be obtained by alternate iterative algorithms.

We provide a general block diagram of the survey (see figure 1). The survey consists of two parts. In part one, we first give the basic definition and notations of tensor in Chapter A. Then we introduce the basic operation of tensor, and the block diagram of the network structure of tensor in Chapter B. Next, we begin to describe tensor decomposition, including several famous decompositions such as the CP (regularization) decomposition, the Tucker decomposition, the Tensor train decomposition and Higher-order singular value decomposition (also known as higher-order tensor decomposition) in Chapter C. In Chapter D, we give a detailed description of tensor train decomposition and the related algorithms. In Chapter E, i.e., the last section of the first part, we summarize the advantages and disadvantages of these decompositions and applications. In part two, we mainly describe tensor application algorithms in machine learning and deep learning. In Chapter A, we introduce the application of structured tensor in data preprocessing including tensor completion and tensor dictionary learning. In Chapter B of this part, we introduce some applications of tensor in classification, including algorithm innovation and data innovation. Then, we illustrate the application of tensor in regression, including tensor regression and multivariate tensor regression, in Chapter C. In the last of part two, we explain the background of the tensor network and discuss its advantages, shortcomings, opportunities and challenges in detail.

II. PART ONE: TENSOR AND TENSOR OPERATION

A. TENSOR NOTATIONS

A tensor can be seen as a generalization of multidimensional arrays. For example, a scalar quantity can be considered as a

0-order tensor, a vector can be treated as a first-order tensor, and a matrix can be regarded as a second-order tensor. And a third-order tensor looks like a cuboid (see figure 2).

FIGURE 2. A 3rd-order tensor looks like a cuboid [3].

A fourth-order tensor is an extension of the third-order tensor along one dimension (see figure 3).

FIGURE 3. A 4th-order tensor extending along the lateral direction [3].

As you can imagine, a fifth-order tensor is an extension of a third order tensor in two directions (see figure 4).

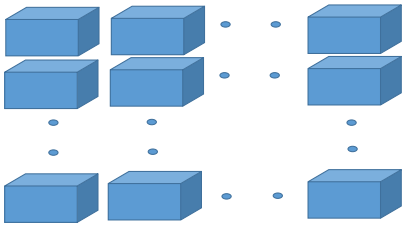

FIGURE 4. A fifth-order tensor extending along the lateral and longitudinal directions [3].

We use underlined uppercase letters to indicate tensors, that is, $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, to represent an Nth-order tensor. Then, similar to the diagonal matrix, we define the diagonal tensor as follows: $\underline{\Lambda} \in R^{I \times I \times I \times \cdots I}$ or $\underline{\Upsilon} \in R^{I \times I \times I \times \cdots I}$. Similar to the transposition of the matrix, we define the transposition of the tensor as follows: $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, $\underline{Y}^T \in R^{I_N \times I_{N-1} \times I_{N-2} \times \cdots I_1}$. We can also use this symbol to represent 2nd-order tensor (matrix) and 1st-order tensor (vector). For convenience, we use separate symbols to represent 2nd-order tensor (matrix) and 1st-order tensor (vector) respectively. We use $Y \in R^{I \times J}$ to represent matrix, $\boldsymbol{y} \in R^I$ for vector, $y \in R^0$ for scalar.

We use $y_{i_1, i_2, \cdots, i_N}$ to represent the entries of an Nth-order tensor $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$. The order of a tensor is the total number of its "dimension" or "mode". The size of a tensor means the range of values that can be obtained for a dimension of tensor. For example, a tensor $\underline{Y} \in R^{3 \times 4 \times 5 \times 6}$ is of order 4, size 3 in mode-1, size 4 in mode-2, size 5 in mode-3 and size 6 in mode-4. In order to describe the tensor more simply, simple tensor network diagram will be used. We use the geometric nodes of a square(sometimes with polygons such as pentagons or hexagons) to represent tensor, and the outgoing line of the node represents the index of a particular dimension (see figure 5 and figure 6). The Nth-order tensor can be expressed in a similar way.

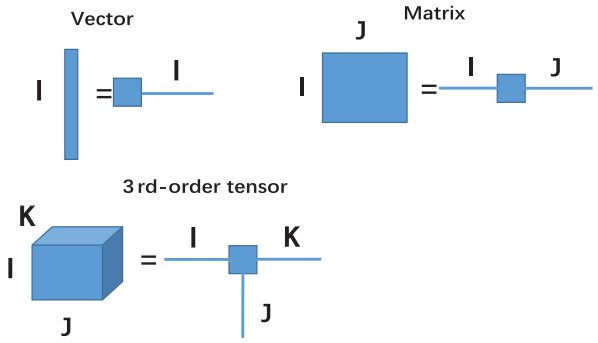

FIGURE 5. A simple network diagram of the tensor, vector $y \in R^I$, matrix $Y \in R^{I \times J}$, 3rd-order tensor $\underline{Y} \in R^{I \times J \times K}$ [3].

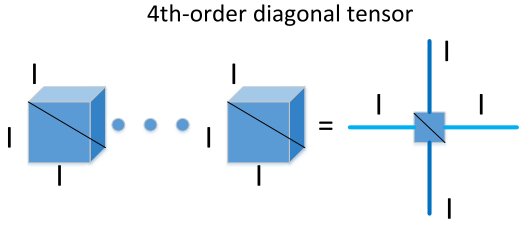

FIGURE 6. A simple network diagram of the 4th-order diagonal tensor, $\underline{\Upsilon} \in R^{I \times I \times I \times I}$.

We need to know the definition of tensor slice and tensor fiber. Tensor fiber(see figure 7) is a vector equivalent to fixing two tensor indices, and tensor slice(see figure 8) is a matrix equivalent to fixing one indices. We use a simple example to

FIGURE 7. Tensor fibers (vectors) for a 3rd-order tensor. It's like tofu being cut in both directions [127].

FIGURE 8. Tensor slices (matrices) for a 3rd-order tensor. It's like tofu being cut in one direction [127].

illustrate tensor slice and tensor fiber.

$$\underline{C} = \left[\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}\right] \tag{1}$$

This is a 3rd-order tensor $\underline{C} \in R^{2\times2\times2}$. For tensor slices(matrices), we can get two matrices $\underline{C}(1,:,:)$ and $\underline{C}(2,:,:)$ when we fix the first dimension: $\left[\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 5 & 6 \\ 7 & 8 \end{smallmatrix}\right]$, which we usually call them horizontal slices. If we fix the second dimension, we can get another two matrices $\underline{C}(:,1,:)$ and $\underline{C}(:,2,:)$: $\left[\begin{smallmatrix} 1 & 2 \\ 5 & 6 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 3 & 4 \\ 7 & 8 \end{smallmatrix}\right]$, which we usually call them lateral slices. If we fix the third dimension, we can still get another two matrices $\underline{C}(:,:,1)$ and $\underline{C}(:,:,2)$: $\left[\begin{smallmatrix} 1 & 3 \\ 5 & 7 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 2 & 4 \\ 6 & 8 \end{smallmatrix}\right]$, which we usually call them frontal slices.

For tensor fibers(vectors), we can get four vectors $\underline{C}(1,1,:)$, $\underline{C}(1,2,:)$, $\underline{C}(2,1,:)$, $\underline{C}(2,2,:)$ when we fix the first and the second indices: [1 2], [3 4], [5 6], [7 8]. If we fix the first and the third indices, we can get another four vectors $\underline{C}(1,:,1)$, $\underline{C}(1,:,2)$, $\underline{C}(2,:,1)$, $\underline{C}(2,:,2)$: [1 3], [2 4], [5 7], [6 8]. If we fix the second and the third indices, we can get another four vectors $\underline{C}(:,1,1)$, $\underline{C}(:,1,2)$, $\underline{C}(:,2,1)$, $\underline{C}(:,2,2)$: [1 5], [2 6], [3 7], [4 8].

B. TENSOR OPERATION

In this chapter, we begin to discuss some basic tensor operations. Tensor operations are similar to traditional linear algebras, but are richer and more meaningful than them. The same operation will also be applied to the following tensor decomposition in Chapter C. In order to get a clearer description of the formulas, we will give examples and graphical instructions. Thirteen tensor calculation formulas will be given. We first give the definition of the following operations: $\bigotimes$ means the Kronecker product, $\odot$ means the Khatri-Rao product, $\circ$ means the outer product, $\times_n$ means the mode-n product. Next, we introduce a few commonly used formulas.

1) THE SUM OF TWO TENSORS

$$\underline{C} = \underline{A} + \underline{B} \tag{2}$$

where $\underline{A} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$, $\underline{B} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$, $\underline{C} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$, $c_{i_1,\cdots,i_N} = a_{i_1,\cdots,i_N} + b_{i_1,\cdots,i_N}$.

2) THE MODE-n PRODUCT OF A TENSOR AND A VECTOR

$$\underline{C} = \underline{A} \times_{nv} \boldsymbol{b} \tag{3}$$

where in $\times_{nv}$, v means vector, n means mode-n, $\underline{A} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$ means the Nth-order tensor, and $\boldsymbol{b} \in R^{I_n}$ means the vector. They yield a tensor $\underline{C} \in R^{I_1\times\cdots\times I_{n-1}\times I_{n+1}\times\cdots\times I_N}$ with entries $c_{i_1,\cdots,i_{n-1},i_{n+1},\cdots,i_N} = \sum_{i_n=1}^{I_n} a_{i_1,\cdots,i_{n-1},i_n,i_{n+1},\cdots,i_N} b_{i_n}$. For example,

$$\underline{C} = \left[\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}\right] \times \begin{bmatrix} 2 \\ 3 \end{bmatrix} \tag{4}$$

where $\underline{C}_{11} = \underline{A}_{111}b_1 + \underline{A}_{121}b_2 = 1\times2 + 3\times3 = 11$. And so on, we can get $\underline{C} = \left[\begin{smallmatrix} 11 & 16 \\ 31 & 36 \end{smallmatrix}\right]$.

3) THE MODE-n PRODUCT OF A TENSOR AND A MATRIX

$$\underline{C} = \underline{A} \times_{nm} B \tag{5}$$

where in $\times_{nm}$, m means matrix, n means mode-n, $\underline{A} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$ means the Nth-order tensor, $B \in R^{J\times I_n}$ means the matrix. They yield a tensor $\underline{C} \in R^{I_1\times\cdots\times I_{n-1}\times J\times I_{n+1}\times\cdots\times I_N}$ with entries $c_{i_1,\cdots,i_{n-1},j,i_{n+1},\cdots,i_N} = \sum_{i_n=1}^{I_n} a_{i_1,\cdots,i_{n-1},i_n,i_{n+1},\cdots,i_N} b_{j,i_n}$.

4) THE MODE-(a,b) PRODUCT(TENSOR CONTRACTION) OF A TENSOR AND ANOTHER TENSOR

$$\underline{C} = \underline{A} \times_{(a,b)} \underline{B} \tag{6}$$

where $\underline{A} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$ means the Nth-order tensor, $\underline{B} \in R^{J_1\times J_2\times J_3\times\cdots J_M}$ means another tensor and here we should note that $I_a = J_b$ ($a \in [1,N], b \in [1,M]$). They yield a tensor $\underline{C} \in R^{I_1\times\cdots\times I_{a-1}\times I_{a+1}\times\cdots I_N\times J_1\times\cdots J_{b-1}\times J_{b+1}\cdots\times J_M}$ with entries $c_{i_1,\cdots,i_{a-1},i_{a+1},\cdots,i_N,j_1,\cdots,j_{b-1},j_{b+1},\cdots,j_M} = \sum_{i_a=1}^{I_a} a_{i_1,\cdots,i_a,\cdots,i_N} b_{j_1,\cdots,j_{b-1},i_a,j_{b+1},\cdots,j_M}$. Note that it is also called tensor contraction because the dimension of the new tensor is the sum of the dimensions of the original two tensors minus the dimension of the same size. We draw a picture to show tensor contraction(see figure 9). For convenience, when two tensors have same size of one dimension, the () in the above formula is usually omitted, that is $\underline{C} = \underline{A} \times_{a,b} \underline{B}$.

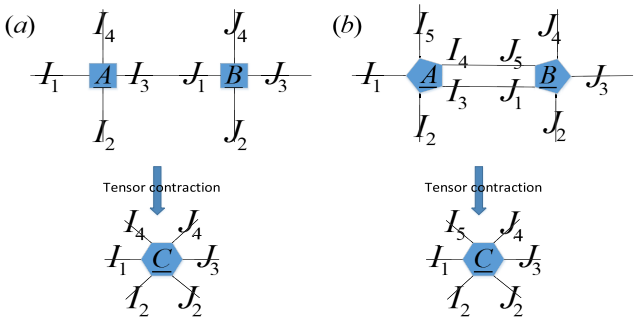

FIGURE 9. (a) the tensor contraction of two 4th-order tensors, $I_3 = J_1$, $\underline{C} = \underline{A} \times_{(3,1)} \underline{B} \in R^{I_1\times I_2\times I_4\times J_2\times J_3\times J_4}$. (b) the tensor contraction of two 5th-order tensors, $I_3 = J_1$, $I_4 = J_5$, $\underline{C} = \underline{A} \times_{(3,1)(4,5)} \underline{B} \in R^{I_1\times I_2\times I_5\times J_2\times J_3\times J_4}$.

From the three formulas (the mode-n product of tenor and vector, matrix, tensor) above, we can see that the mode-n product offsets the same dimension, and the different dimensions are added, which is similar to the product of the matrix, but a little different.

5) THE TRANSPOSITION OF TENSOR CONTRACTION

$$\underline{C} = (\underline{A} \times_{a,b} \underline{B})^T = \underline{B}^T \times_{b,a} \underline{A}^T \tag{7}$$

6) THE CONJUGATE TRANSPOSE OF A 3RD-ORDER TENSOR

The conjugate transpose of a 3rd-order tensor $\underline{X} \in R^{I_1 \times I_2 \times I_3}$ is a tensor $\underline{X}^* \in R^{I_2 \times I_1 \times I_3}$ obtained by conjugate transposing each of the frontal slices(fix the third order $\underline{X}(:,:,i_3)$) and then reversing the order of transposed frontal slices 2 through I_3. We give a simple example to show(see formula 8):

$$\begin{aligned} C &= \left[\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}\right] \\ C^* &= \left[\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}, \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}\right] \end{aligned} \tag{8}$$

7) THE OUTER PRODUCT OF A TENSOR AND ANOTHER TENSOR

$$\underline{C} = \underline{A} \circ \underline{B} \tag{9}$$

where $\underline{A} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$ and $\underline{B} \in R^{J_1 \times J_2 \times J_3 \times \cdots J_M}$. They yield an (N+M)th-order tensor $\underline{C}$ with entries $c_{i_1,\cdots,i_N,j_1,\cdots,j_M} = a_{i_1,\cdots,i_N} b_{j_1,\cdots,j_M}$.

8) THE (RIGHT)KRONECKER PRODUCT OF TWO TENSORS

$$\underline{C} = \underline{A} \otimes_R \underline{B} \tag{10}$$

where in $\otimes_R$, R means right, $\underline{A} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$ and $\underline{B} \in R^{J_1 \times J_2 \times J_3 \times \cdots J_N}$. They yield a tensor $\underline{C} \in R^{J_1 I_1 \times \cdots \times J_N I_N}$ with entries $c_{\overline{i_1 j_1},\cdots,\overline{i_N j_N}} = a_{i_1,\cdots,i_N} b_{j_1,\cdots,j_N}$, where $\overline{i_N j_N} = j_N + (i_N - 1)J_N$ is called multi-indices. Note that for Kroneker product, two tensors must have the same dimension. They must not carry out the Kronecker product of the matrix and the 3rd-order tensor, and must have a 3rd-order tensor and another 3rd-order tensor. A simple example of a second-order matrix is provided, as follows:

$$\begin{aligned} C &= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes_R \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \\ &= \begin{bmatrix} 1\times5 & 1\times6 & 2\times5 & 2\times6 \\ 1\times7 & 1\times8 & 2\times7 & 2\times8 \\ 3\times5 & 3\times6 & 4\times5 & 4\times6 \\ 3\times7 & 3\times8 & 4\times7 & 4\times8 \end{bmatrix} \end{aligned} \tag{11}$$

In fact,

$$\underline{C} = \underline{A} \otimes_R \underline{B} = \underline{B} \otimes_L \underline{A} \tag{12}$$

The right-most equation is called the left Kronecker product.

9) THE RIGHT KHATRI-RAO PRODUCT OF MATRICES

$$\begin{aligned} C &= A \odot_R B \\ &= [a_1 \otimes_R b_1, a_2 \otimes_R b_2, \cdots, a_K \otimes_R b_K] \in R^{IJ \times K} \end{aligned} \tag{13}$$

where $A = [a_1, a_2, a_3, \cdots, a_K] \in R^{I \times K}, B = [b_1, b_2, b_3, \cdots, b_K] \in R^{J \times K}$. The left Khatri-Rao product of matrices is similar. For convenience, the right Khatri-Rao product of matrices is used in this survey, so we will abbreviate the right Khatri-Rao product of matrices $\otimes_R$ as $\otimes$.

10) THE MODE-n MATRICIZATION AND VECTORIZATION OF THE TENSOR

In the previous section, we introduced the concepts of tensor slice and tensor fiber. Here we present two similar but different concepts, matricization and vectorization. Tensor slice and tensor fiber take some specific elements of the tensor to form a matrix or a vector, while matricization and vectorization are to matricize or vectorize all the elements. We now give a formal definition.

The mode-n matricization of a tensor, $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, is as follows:

$$\begin{aligned} mat(Y)_n &= mat(\underline{Y})_n = \underline{Y}_{mn} \in R^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N} \\ &\quad Where\ for\ the\ matrix\ element\ (i_n j), \\ j &= 1 + \sum_{k=1, k\neq n}^{N} (i_k - 1) J_k\ with\ J_k = \prod_{m=1, m\neq n}^{k-1} I_m \end{aligned} \tag{14}$$

The mode-n vectorization of a tensor, $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, is as follows:

$$vec(\underline{Y})_n = \underline{Y}_{vn} \in R^{I_n I_1 \cdots I_{n-1} I_{n+1} \cdots I_N} \tag{15}$$

For the mode-n vectorization, we first perform mode-n matricization and then stack the matrix in columns. Of course, vectors and matrices can also be transformed into tensor. We give examples of mode-1 matricization and vectorization of a 3rd-order tensor(see formula 16).

$$C = \left[\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}\right] \Leftrightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 1 \\ 5 \\ 2 \\ 6 \\ 3 \\ 7 \\ 4 \\ 8 \end{bmatrix} \tag{16}$$

11) THE TENSOR QUANTITATIVE PRODUCT

$$c = \underline{A} \bullet \underline{B} = \sum_{j_1=1}^{J_1} \cdots \sum_{j_N=1}^{J_N} a_{j_1,\cdots,j_N} b_{j_1,\cdots,j_N} \tag{17}$$

where $\underline{A} \in R^{J_1 \times \cdots \times J_N}, \underline{B} \in R^{J_1 \times \cdots \times J_N}$. Note that the requirements for tensor quantitative product are too strict. Not only the dimension of the two tensors should be the same, but also the size of the two tensors has to be the same. In this way, **we can further define the Frobenius norm of tensor.**

$$\|\underline{A}\|_F = (\underline{A} \bullet \underline{A})^{1/2} \tag{18}$$

12) THE TENSOR ELEMENT PRODUCT

$$\underline{C} = \underline{A} \circledast \underline{B} \tag{19}$$

where $\underline{C} \in R^{I_1 \times \cdots \times I_N}, \underline{A} \in R^{I_1 \times \cdots \times I_N}, \underline{B} \in R^{I_1 \times \cdots \times I_N}$, $c_{i_1,\cdots,i_N} = a_{i_1,\cdots,i_N} b_{i_1,\cdots,i_N}$. Note that since it is element

product, similar to the quantitative product, the two tensors must have the same dimension and the same size.

13) THE TENSOR TRACE

Similar to trace of the matrix, tensor also has a trace. (Gu, 2009) [162] proposed the concept of tensor trace. Let's first look at the concept of inner indices. If a tensor has the same size for several dimensions, those same size dimensions are called inner indices. For example, a tensor $\underline{X} \in R^{A\times B\times A}$ has two inner indices. Modes 1 and 3 are both size A. Then, we define the following concept of tensor trace:

$$x = Trace(\underline{X}) = \sum_{r=1}^{R} \underline{X}(r,:,r) \tag{20}$$

$$x = [x_1, x_2, \cdots, x_B]^T, \quad x_i = \sum_{r=1}^{R} \underline{X}(r,i,r) \tag{21}$$

$$x = [tr(X_1), \cdots, tr(X_B)]^T, \quad X_i \in R^{R\times R} \tag{22}$$

Let's give an example of the 3rd-order tensor that we have used before.

$$\underline{C} = \left[\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}, \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix}\right] \tag{23}$$

$$c = Trace(\underline{C}) = [1+6, 3+8]^T = [7, 11]^T \tag{24}$$

$$C_1 = \begin{bmatrix}1 & 2\\ 5 & 6\end{bmatrix}, \quad C_2 = \begin{bmatrix}3 & 4\\ 7 & 8\end{bmatrix} \tag{25}$$

14) THE TENSOR CONVOLUTION

Tensor also has convolution, which is similar to matrix convolution. For two Nth-order tensors $\underline{A} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$ and $\underline{B} \in R^{J_1\times J_2\times J_3\times\cdots J_N}$. Their tensor convolution is as follows:

$$\underline{C} = \underline{A} * \underline{B} \tag{26}$$

$\underline{C} \in R^{(I_1+J_1-1)\times(I_2+J_2-1)\times\cdots\times(I_N+J_N-1)}$, with entries $c_{k_1,k_2,\cdots,k_N} = \sum_{j_1=1}^{J_1}\sum_{j_2=1}^{J_2}\cdots\sum_{j_N=1}^{J_N} b_{j_1,\cdots,j_n} a_{k_1-j_1,\cdots,k_n-j_n}$ For a simple and intuitive display, we use matrix convolution to illustrate (see figure 10).

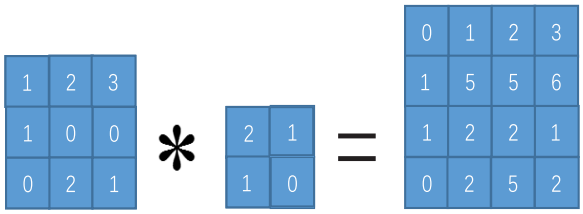

FIGURE 10. A schematic diagram of the results of matrix convolution, with $C_{11} = 0\times 1 = 0$, $C_{12} = 1\times 1 + 0\times 2 = 1$, $C_{22} = 1\times 2 + 2\times 1 + 1\times 1 + 0\times 0 = 5, \cdots$.

15) SHORT SUMMARY

The formulas for tensor operations described above are relatively basic ones. Because tensor can be seen as a generalization of matrices and vectors, the above formulas also apply to vectors and matrices (just change the dimension to 1 or 2 in the formulas). Many researchers have also defined some new operations, such as the strong Kronecker product(de Launey and Seberry [140]; Phan *et al.* [8]) and the mode-n Khatri-Rao product of tensors (Ballard *et al.*) [33]. Based on the Kroneker product, these two operations are just grouped into blocks to perform the Kroneker product operation.

This chapter mainly introduces basic calculation formulas commonly used by tensor. If you want to know more about many other formulas, please refer to (Kolda and Bader) [127].

C. TENSOR DECOMPOSITION

This chapter begins to discuss the knowledge of tensor decomposition, which is similar but different from matrix decomposition. Tensor decomposition aims to reduce the computational complexity while ensuring the data structure, so as to better deal with the data. Tensor decomposition technology has been gradually used in data analysis and processing. This chapter will focus on five main types of decomposition, i.e., the Canonical Polyadic(CP) decomposition, the Tucker decomposition, the MultiLinear Singular Value(the higher-order SVD or HOSVD) decomposition, the Hierarchical Tucker(HT) decomposition and the tensor-train(TT) decomposition, respectively.

1) THE CANONICAL POLYADIC(CP) DECOMPOSITION

Before introducing CP decomposition, we first introduce the bidirectional component analysis, i.e., **the constrained low-rank matrix factorization**.

$$C = \Lambda A B^T + E = \sum_{r=1}^{R} \lambda_r \boldsymbol{a}_r \boldsymbol{b}_r^T + E \tag{27}$$

where $\Lambda = diag(\lambda_1, \cdots, \lambda_r)$ is an diagonal matrix. $C \in R^{I\times J}$ is a known matrix (for example, known input data, etc.). $E \in R^{I\times J}$ is a noise matrix. $A = [\boldsymbol{a_1}, \cdots, \boldsymbol{a_R}] \in R^{I\times R}$, $B = [\boldsymbol{b_1}, \cdots, \boldsymbol{b_R}] \in R^{J\times R}$ are two unknown factor matrices with $\boldsymbol{a}_r \in R^I$, $\boldsymbol{b}_r \in R^J$, $r \in [1, R]$. In fact, if the noise matrix is very small, it can be ignored and the upper expression can be written as $C \approx \Lambda A B^T$.

In fact, based on low rank matrix decomposition, (Hitchcock [31]; Harshman, 1970 [110]) proposed the CP decomposition of tensor. Before introducing the definition of CP decomposition, **we give the definition of rank-1 tensor**. If a tensor can be represented as follows:

$$\underline{Y} = \boldsymbol{b}^1 \circ \boldsymbol{b}^2 \circ \cdots \boldsymbol{b}^N \tag{28}$$

where $\underline{Y} \in R^{I_1\times I_2\times I_3\times\cdots I_N}$, $\boldsymbol{b}^n \in R^{I_n}$, $y_{i_1,\cdots,i_N} = b^1_{i_1}\cdots b^N_{i_N}$, then we call the tensor rank-1 tensor. In CP decomposition, tensor is decomposed into the linear sum of these vectors. **CP decomposition is defined as follows**:

$$\underline{Y} \approx \sum_{r=1}^{R} \lambda_r \boldsymbol{b}_r^1 \circ \boldsymbol{b}_r^2 \circ \cdots \boldsymbol{b}_r^N = \underline{\Lambda} \times_{1m} B_1 \times_{2m} B_2 \cdots \times_{Nm} B_N \tag{29}$$

Similar to the constrained low-rank matrix factorization that we have just described, where $\lambda_r = \Lambda_{r,r,r,\cdots,r}$, $r \in [1, R]$ are entries of the diagonal core tensor $\underline{\Lambda} \in R^{R\times R\times R\times\cdots R}$. $B_n = [\boldsymbol{b}_1^n, \boldsymbol{b}_2^n, \cdots, \boldsymbol{b}_R^n] \in R^{I_n\times R}$ are factor matrices. With the help of other formulas, CP decomposition has a lot of other similar expressions, among which we give two commonly used equations. Considering a special case, when all factor matrices are the same, we call the CP decomposition **a symmetric tensor decomposition**, then $\underline{Y} \in R^{I\times I\times I\cdots I}$. Figure 11 shows CP decomposition of a 3rd-order tensor(see figure 11).

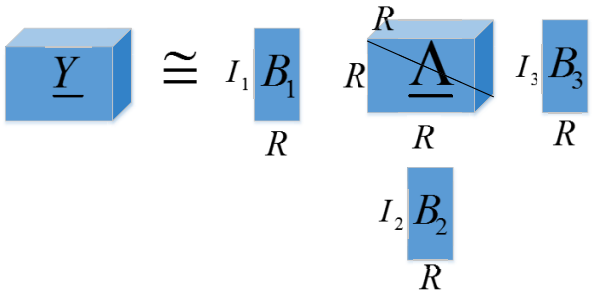

FIGURE 11. CP decomposition of a 3rd-order tensor, $\underline{Y} \approx \underline{\Lambda} \times_{1m} \boldsymbol{B}_1 \times_{2m} \boldsymbol{B}_2 \times_{3m} \boldsymbol{B}_3$ [3].

CP Rank: Similar to matrix, tensor also has a rank. Since it is a CP decomposition at this time, we call it CP rank. CP rank refers to the smallest R for which the CP decomposition in the above formula holds exactly. We use $r_{cp}(\underline{Y})$ to represent the CP rank.

In practice, unlike traditional matrix decomposition, tensor usually have interference (such as noise or even data loss). Therefore, it is usually difficult to find the exact solution of CP decomposition, so most of them are approximate solutions.

So the question comes, that how do we get tensor approximate CP decomposition, or in other words, that how can we get the core tensor? The general approach is to first find the factor matrix B_n by minimizing an appropriate loss function. (A.Vorobyov, 2005) [116] presents a loss function similar to the least square method.

$$J(B_1, \cdots, B_N) = \|\underline{Y} - \underline{\Lambda} \times_{1m} B_1 \cdots \times_{Nm} B_N\|_F^2 \quad (30)$$

Our goal is to minimize the loss function in the upper form, and we use the alternating least square method, which means iterative optimization by fixing the value of a variable other than one. That is to say, one of those N factor matrices B_n, is optimized separately at a time, keep the values of other N-1 factor matrices unchanged (we first initialize all N factor matrices, and optimize only B_1 by gradient descent while keep the initial values of B_2 to B_N unchanged). This becomes a single variable loss function optimization problem. Then it continues to iterate until the iteration threshold is reached or the algorithm has converged. The derivation is not given here. We give the results directly, and take the 4th-order tensor as an example to write the following example: the factor matrices can be iterative updated as

$$B_n = \underline{Y}_{mn}[(B_N \odot_R \cdots B_{n+1} \odot_R B_{n-1} \cdots B_1)^T]^\dagger \quad (31)$$

where $\underline{Y}_{mn}$ represents the mode-n matricization of tensor $\underline{Y}$, $\dagger$ means the Moore-Penrose pseudo-inverse of the matrix. We give an algorithm for the 4th-order tensor CP decomposition (see Algorithm 1).

Algorithm 1 The CP Decomposition Algorithm of a 4th-Order Tensor

Input:
The 4th-order tensor $\underline{Y} \in R^{I\times J\times K\times L}$

Output:
Factor matrices A,B,C,D and the core tensor $\underline{\Lambda}$

Initialize A,B,C,D and CP rank R, where $R \leq min\{IJ, JK, IK\}$;
while the iteration threshold does not reach or the algorithm has not converged **do**
  $A = \underline{Y}_{m1}[(D \odot_R C \odot_R B)^T]^\dagger$;
  Normalize column vectors of A to unit vector;
  $B = \underline{Y}_{m2}[(D \odot_R C \odot_R A)^T]^\dagger$;
  Normalize column vectors of B to unit vector;
  $C = \underline{Y}_{m3}[(D \odot_R B \odot_R A)^T]^\dagger$;
  Normalize column vectors of C to unit vector;
  $D = \underline{Y}_{m4}[(C \odot_R B \odot_R A)^T]^\dagger$;
  Normalize column vectors of D to unit vector;
  Save the value of the norms of the R column vectors in the factor matrix C to the core tensor $\underline{\Lambda}$;
end while
return Factor matrices A,B,C,D and the core tensor $\underline{\Lambda}$

From the above algorithm, we can see that the key to calculate CP decomposition is to calculate Khatri-Rao product and the pseudo inverse of the matrices. (Choi and Vishwanathan [63]; Karlsson *et al.* [79]) proposed the least-squares solution method of CP decomposition and the detailed derivation process can be referenced by them.

2) THE TUCKER DECOMPOSITION

The Tucker decomposition was first proposed by (Tucker) [81], so it was named Tucker decomposition. Similar to the CP decomposition, the Tucker decomposition also divides tensor into small size of core tensor and factor matrices, but what we need to pay attention to is that the core tensor here is not necessarily the diagonal tensor. **We define the Tucker decomposition as follows**:

$$\begin{aligned}\underline{Y} &\approx \sum_{r_1=1}^{R_1}\cdots\sum_{rN=1}^{R_N} a_{r_1 r_2\cdots r_N}\boldsymbol{b}_{r_1}^1 \circ \boldsymbol{b}_{r_2}^2 \circ \cdots \boldsymbol{b}_{r_N}^N \\ &= \underline{A} \times_{1m} B_1 \times_{2m} B_2 \cdots \times_{Nm} B_N \\ \underline{Y}_{v1} &= (B_N \otimes_R B_{N-1} \cdots \otimes_R B_1)\underline{A}_{v1} \end{aligned} \quad (32)$$

where $a_{r_1 r_2\cdots r_N}$ are entries of the small size core tensor $\underline{A} \in R^{R_1\times R_2\cdots R_N}$, $B_n = [\boldsymbol{b}_1^n, \boldsymbol{b}_2^n, \cdots, \boldsymbol{b}_{R_n}^n] \in R^{I_n\times R_n}$ are factor matrices, $\underline{Y}_{v1}$ is the mode-1 vectorization of the tensor $\underline{Y}$,

and $\underline{A}_{v1}$ is the mode-1 vectorization of the core tensor $\underline{A}$. In fact, CP decomposition is a special form of Tucker decomposition. We draw two decomposition figures to compare them intuitively (see figure 12 and figure 13).

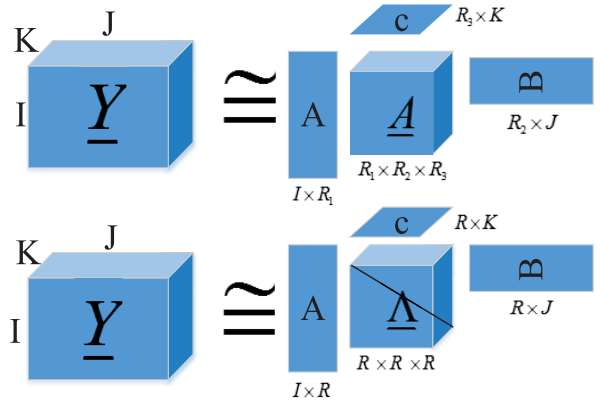

FIGURE 12. Comparison of CP decomposition and Tucker decomposition for a 3rd-order tensor. The above figure is Tucker decomposition $\underline{Y} \approx \underline{A} \times_{1m} A \times_{2m} B \times_{3m} C$ and the following figure is CP decomposition $\underline{Y} \approx \underline{\Lambda} \times_{1m} A \times_{2m} B \times_{3m} C$, $\underline{Y} \in R^{I \times J \times K}$.

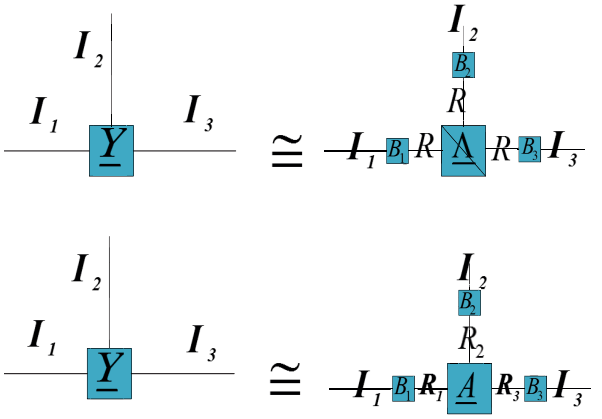

FIGURE 13. Comparison of CP decomposition and Tucker decomposition for a 3rd-order tensor, (simple tensor network schematic mode). The above figure is CP decomposition $\underline{Y} \approx \underline{\Lambda} \times_{1m} B_1 \times_{2m} B_2 \times_{3m} B_3$ and the following figure is Tucker decomposition $\underline{Y} \approx \underline{A} \times_{1m} B_1 \times_{2m} B_2 \times_{3m} B_3$, $\underline{Y} \in R^{I_1 \times I_2 \times I_3}$. Note that the factor matrices can be represented by different English letters.

As we can see from the two figures (figure 12 and figure 13), the CP decomposition is a special form of Tucker decomposition. Once the normal core tensor degenerates into a diagonal core tensor, the Tucker decomposition becomes the CP decomposition. As did to CP decomposition, we can use the properties of other formulas to represent the Tucker decomposition. Here we give two that commonly used.

Multiple Linear Rank (Tucker Rank): Unlike CP decomposition, a new rank is redefined here for Tucker decomposition, which we call multiple linear rank. The multiple linear rank of the tensor is $(R_1, R_2, \cdots, R_N)$. Moreover, if the upper Tucker decomposition can get an equal sign, then the multiple linear rank of a tensor $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$ is defined as follows:

$$r_{ml}(\underline{Y}) = (r(\underline{Y}_{m1}), r(\underline{Y}_{m2}), \cdots, r(\underline{Y}_{mN})) \tag{33}$$

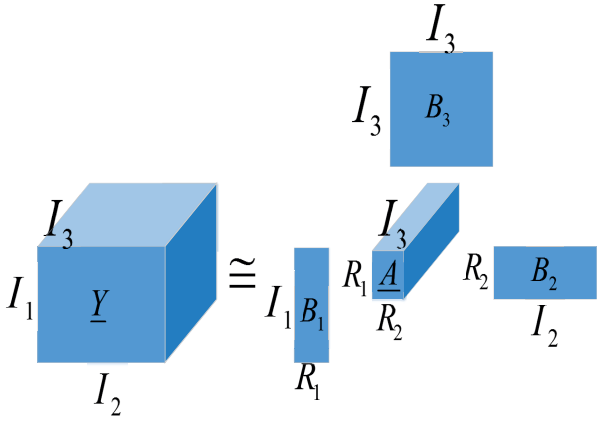

FIGURE 14. Because only B_3 is the identity matrix, the graph is the Tucker-1 decomposition model.

where $\underline{Y}_{mn}$ is the mode-n matricization of tensor $\underline{Y}$, $r(\underline{Y}_{mn})$ means the matrix rank of the mode-n matricization of tensor $\underline{Y}$.

If the upper Tucker decomposition can get an equal sign, then it will have the following important properties:

1. The CP rank of any tensor $\underline{Y} = \underline{A} \times_{1m} B_1 \times_{2m} B_2 \cdots \times_{Nm} B_N$ is equal to the small size core tensor $\underline{A}$.

$$r_{cp}(\underline{Y}) = r_{cp}(\underline{A}) \tag{34}$$

where $\underline{A}$ is the small size core tensor of $\underline{Y}$.

2. If a tensor $\underline{Y}$ has full column rank factor matrices and its multiple linear rank=$(R_1, R_2, \cdots, R_N)$, then

$$R_n \leq \prod_{k \neq n}^{N} R_k, \forall n. \tag{35}$$

3. If a tensor $\underline{Y} \in R^{I \times I \times I \cdots \times I}$ has full column rank factor matrices and its corresponding CP decomposition is symmetric (all the factor matrices are the same), then its core tensor, $\underline{A} \in R^{R \times R \times R \cdots R}$, is also symmetric. In this case, Tucker decomposition is equivalent to CP decomposition, which is called symmetric decomposition (as we've defined before.).

4. If a tensor $\underline{Y} \in R^{I_1 \times I_2 \times \cdots I_N}$ has full column rank factor matrices and all the factor matrices are orthogonal, then the Frobenius norms of the tensor $\underline{Y}$ and it's core tensor $\underline{A}$ are equal.

$$\|\underline{Y}\|_F = \|\underline{A}\|_F \tag{36}$$

There are also some special Tucker decompositions, which are briefly described here. The full column rank of all the factor matrices in the property 3 we just have is usually called **an independent Tucker decomposition**. On this basis, if all the factor matrices are also orthogonal matrices, that is $B_n^T B_n = I_{R_n}$, we call it **orthogonal Tucker decomposition**. If there are N identity matrices in factor matrices, we usually call them **Tucker-N decomposition** (see figure 14). For example, $\underline{Y} = \underline{A} \times_{1m} B_1 \times_{2m} B_2 \times_{3m} I \times_{4m} I$, then we call it Tucker-2 decomposition.

Here we briefly introduce some operational properties of Tucker decomposition. Consider two Nth-order tensors $\underline{X} = \underline{A}_X \times_{1m} B_1 \times_{2m} B_2 \cdots \times_{Nm} B_N$, $\underline{Y} = \underline{A}_Y \times_{1m} C_1 \times_{2m} C_2 \cdots \times_{Nm} C_N$, their multiple linear rank is $(R_1, R_2, \cdots, R_N)$ and $(Q_1, Q_2, \cdots, Q_N)$, respectively. Then, they will have the following computational properties:

1. The (Right or left) Kronecker product of the two tensors:

$$\begin{aligned}\underline{Z} &= \underline{X} \otimes \underline{Y} \\ &= (\underline{A}_X \otimes \underline{A}_Y) \times_{1m} (B_1 \otimes C_1) \cdots \times_{Nm} (B_N \otimes C_N)\end{aligned} \quad (37)$$

2. The Hadamard product of the two tensors (the same sizes and order):

$$\underline{Z} = \underline{X} \circledast \underline{Y} = (\underline{A}_X \otimes_L \underline{A}_Y) \times_{1m} (B_1 \odot C_1) \cdots \times_{1m} (B_N \odot C_N) \quad (38)$$

3. The inner product of the two tensors:

$$\begin{aligned}z &= \underline{X} \bullet \underline{Y} = (vec(\underline{X})_1)^T vec(\underline{Y})_1 \\ &= (vec(\underline{A}_X)_1)^T \otimes_L ((B_1)^T C_1) \otimes_L ((B_2)^T C_2) \cdots \otimes_L \\ &\quad ((B_N)^T C_N) vec(\underline{A}_Y)_1\end{aligned} \quad (39)$$

Here it is noted that we used a vector equivalent representation of the Tucker decomposition:

$$vec(\underline{Y})_1 = (B_N \otimes_R B_{N-1} \cdots \otimes_R B_1) vec(\underline{A})_1 \quad (40)$$

We put the formula 40 in 39 and get the result.

3) THE HIERARCHICAL TUCKER DECOMPOSITION

(Hackbusch and Khn [142]; Grasedyck [78]) produced the Hierarchical Tucker decomposition. The Hierarchical Tucker(HT) decomposition decomposes tensor in a hierarchical way, and it is similar to a binary tree split. It is important to note that for the HT decomposition, all the core tensor must be less than or equal to the third order. In other words, the factor matrices connected to the core tensor cannot exceed 3. Simpler, if you use a tensor network diagram to illustrate, a core tensor can't have more than three lines connected to it. Also, the HT decomposition model graphs cannot contain any loops. We draw a diagram of the HT decomposition of 5th-order tensor and 6th-order tensor so that we can understand it more intuitively (see figure 15 and figure 16).

From the figure 15 and the figure 16, we can find that the first step of HT decomposition is to extract the dimensions to be decomposed. For a 5th-order tensor, we can extract any one dimension or any two dimensions and the steps are repeated until the 5th-order tensor becomes five factor matrices. In fact, we can discover that the HT decomposition replaces the core tensor $\underline{A}$ of Tucker decomposition, with low-order interconnected kernels, thus forming a distributed tensor network. We draw the conversion of HT decomposition and Tucker decomposition of the 5th-order tensor(see figure 17). Of course, we can find that with the increase in dimension, these distributed networks (HT decomposition networks) are not unique(see figure 16).

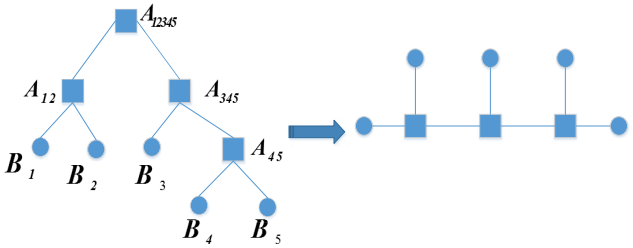

FIGURE 15. Schematic diagram of HT decomposition of 5th-order tensor, in which the core tensor is split into two small-size 3rd-order tensors A_{12}, A_{345}, and the right core tensor is split into the factor matrix B_3 and the 3rd-order core tensor of smaller size A_{45}. Finally, A_{12} and A_{45} continue to be decomposed into the last four factor matrices B_1, B_2, B_3, B_4. The diagram on the right is the HT tensor network structure diagram with the core tensor A_{12345} in the original left image replaced by a connecting line.

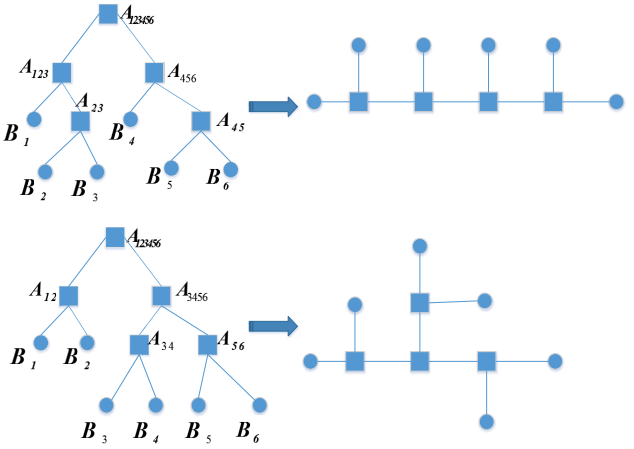

FIGURE 16. Similar to the HT decomposition of the fifth-order tensor in figure 15, it is noted here that since the decomposition of the core tensor is different at the beginning, there are two kinds of decomposition cases. The above figure is to decompose A_{123456} according to dimensions 12 and 3456 respectively, and the following figure is to decompose A_{123456} according to dimensions 123 and 456 respectively. The results are not the same but both the HT decomposition.

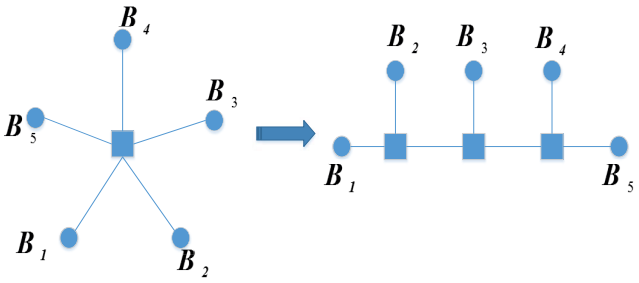

FIGURE 17. The Tucker decomposition of the 5th-order tensor and its equivalent HT decomposition, the right side is the equivalent conversion of the left Tucker decomposition.

We can use the vector form of the Tucker decomposition to explain the HT decomposition network in figure 15.

$$\begin{aligned}vec(\underline{Y})_1 &= (B_1 \otimes_L B_2 \cdots \otimes_L B_5) vec(\underline{A}_{12345}) \\ vec(\underline{A}_{12345})_1 &= vec(\underline{A}_{12})_1 \otimes_L vec(\underline{A}_{345})_1 \\ vec(\underline{A}_{12})_1 &= B_1 \otimes_L B_2 \\ vec(\underline{A}_{345})_1 &= B_3 \otimes_L vec(\underline{A}_{45})_1 \\ vec(\underline{A}_{45})_1 &= B_4 \otimes_L B_5\end{aligned} \quad (41)$$

In fact, the core idea is to replace the core tensor with smaller dimension of tensors until the original tensor is decomposed into factor matrices. Finally, the original tensor is decomposed into a case where several 3rd-order tensors and several factor matrices are connected to each other. Here we introduce the HT decomposition of the 5th-order and the 6th-order tensor. The higher order tensor HT decomposition of the tensor network diagram can be drawn with a similar example and for more details please refer to (Tobler [22]; Kressner *et al.* [23]).

After Tucker decomposition, although the size of the core tensor is reduced, the dimension of the core tensor is still the same as before. When the original tensor dimension is very large (for example, greater than 10), we usually express it with the distributed tensor network similar to the HT decomposition. That is, the dimension of core tensor is not limited to the 3rd order. According to the actually need, it can be 4th or 5th order (see figure 18).

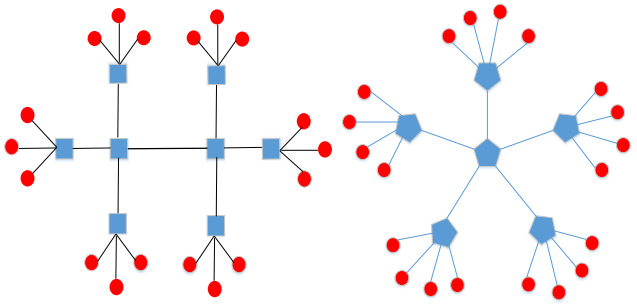

FIGURE 18. The blue rectangles represent the core tensors and the red circles represent the factor matrices. The diagram on the left is an 18th-order tensor HT decomposition tensor network diagram, in which the 4th-order small size core tensors are connected to each other. The diagram on the right is a 20th-order tensor HT decomposition tensor network diagram, in which the 5th-order small size core tensors are connected to each other.

4) THE HIGHER ORDER SVD(HOSVD) DECOMPOSITION

The high-order singular value decomposition of tensor can be considered as another special form of Tucker decomposition (De Lathauwer *et al.*) [73], where the factor matrices and the core tensor are all orthogonal.

The definition of core tensor orthogonality is as follows:

1. The tensor slices in each mode of a tensor should mutually orthogonal, such as, for a 3rd-order tensor $\underline{A} \in R^{I\times J\times K}$

$$\begin{aligned}(A_{a,:,:})(A_{b,:,:}) &= 0, \quad for\ (a \neq b, a, b \in [1, I])\\ (A_{:,c,:})(A_{:,d,:}) &= 0, \quad for\ (c \neq d, c, d \in [1, J])\\ (A_{:,:,e})(A_{:,:,f}) &= 0, \quad for\ (e \neq f, e, f \in [1, K])\end{aligned} \tag{42}$$

2. The Frobenius norms of slices in each mode of a tensor should increase with the increase in the running index, such as, for a 3rd-order tensor

$$\begin{aligned}\|A_{a,:,:}\|_F &\geq \|A_{b,:,:}\|_F, \quad for(a \neq b, a, b \in [1, I])\\ \|A_{:,c,:}\|_F &\geq \|A_{:,d,:}\|_F, \quad for(c \neq d, c, d \in [1, J])\\ \|A_{:,:,e}\|_F &\geq \|A_{:,:,f}\|_F, \quad for(e \neq f, e, f \in [1, K])\end{aligned} \tag{43}$$

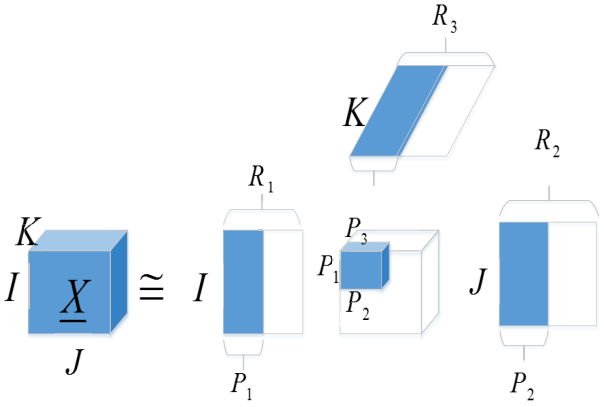

FIGURE 19. The truncated HOSVD decomposition of a 3rd-order tensor [127].

In fact, the orthogonal constraints of tensors and the constraints of matrix SVD decomposition are very similar. Similar to the truncated SVD decomposition of the matrix, the tensor also has a truncated HOSVD decomposition (see figure 19).

The first step in finding the solution of HOSVD decomposition is to first perform the mode-n matricization of the original input tensor and then use a truncated or randomized SVD to find the factor matrices(see equation 157)

$$\underline{X}_{mn} = U_n S_n V_n^T = [U_n^1, U_n^2][S_n^1, 0][V_{n1}^T, V_{n2}^T] \tag{44}$$

When the factor matrix is obtained, the core tensor can be decomposed using the following formula:

$$\underline{A} = \underline{X} \times_{1m} B_1^T \times_{2m} B_2^T \cdots \times_{Nm} B_N^T \tag{45}$$

where $\underline{X} \in R^{I_1\times I_2\cdots I_N}$ is the input tensor, $\underline{A} \in R^{R_1\times R_2\cdots R_N}$ is the core tensor, and $B_n \in R^{I_n\times R_n}$ are the factor matrices. See Algorithm 2 for details and refer to (Vannieuwenhoven *et al.* [101]; Halko *et al.* [96]).

Algorithm 2 The Truncated HOSVD of the Tensor (N.Vannieuwenhoven, 2012) [101]

Input:

The Nth-order input tensor $\underline{X} \in R^{I_1\times I_2\cdots I_N}$ and truncation rank $(R_1, R_2, \cdots, R_N)$ and accuracy ε

Output:

Estimated value $\widehat{\underline{X}} = \underline{A} \times_{1m} B_1 \times_{2m} B_2 \cdots \times_{Nm} B_N$, the core tensor $\underline{A} \in R^{R_1\times R_2\cdots R_N}$ and the factor matrices $B_n \in R^{I_n\times R_n}$ such that $\|\underline{X} - \widehat{\underline{X}}\|_F \leq \varepsilon$

1: $\underline{A} \leftarrow \underline{X}$;
2: **for** n=1 to N **do**
3: $[U_n, S_n, V_n^T] = [U_n^1, U_n^2][S_n^1, 0][V_{n1}^T, V_{n2}^T] = truncated - svd(\underline{A}_{mn}, \frac{\varepsilon}{\sqrt{N}})$;
4: $B_n = U_n^1$;
5: $\underline{A}_{mn} \leftarrow S_n^1 V_{n1}^T$;
6: **end for**
7: $\underline{A} = \underline{X} \times_{1m} B_1^T \times_{2m} B_2^T \cdots \times_{Nm} B_N^T$;
8: **return** the core tensor $\underline{A}$ and factor matrices B_n

After performing the mode-n matricization of the tensor, if the tensor size is too large, we can also obtain the factor matrices by matrix partitioning, as follows:

$$\begin{aligned} \underline{X}_{mn} &= [X_{1n}, X_{2n}, \cdots, X_{Mn}] \\ &= U_n S_n [V_{1n}^T, V_{2n}^T, \cdots, V_{Mn}^T] \end{aligned} \tag{46}$$

where we divide the resulting matrix (called the unfolded matrix) $\underline{X}_{mn}$ into M parts. Then we use the eigenvalue decomposition $\underline{X}_{mn}\underline{X}_{mn}^T = U_n(S_n)^2 U_n^T = \sum_{m=1}^{M} X_{mn}X_{mn}^T$, $U_n = [U_n^1, U_n^2]$, $B^n = U_n^1$. And we can get $V_{mn} = X_{mn}^T U_n (S_n)^{-1}$. Thus, computational complexity and computational memory will be decreased and the efficiency will be improved to some extent by matrix partitioning. At the same time, it also alleviates the curse of dimension problem.

Some researchers proposed a random SVD decomposition algorithm for matrices with large size and low rank. (Halko *et al.*) [96] reduced the original input matrix to a small size matrix by random sketching, i.e., by multiplying a random sampling matrix (see Algorithm 3).

Algorithm 3 The Random SVD Decomposition Algorithm for Large-Size and Low Rank Matrices (Halko *et al.*) [96]

Input:

The large-size and low rank matrix $X \in R^{I\times J}$, estimated rank R, oversampling parameter P, overestimated rank $\widehat{R} = R + P$, exponent of the power method q (q=0 or 1)

Output:

the SVD of X, orthogonal matrix $U \in R^{I\times\widehat{R}}$, diagonal matrix $S \in R^{\widehat{R}\times\widehat{R}}$ and $V \in R^{J\times\widehat{R}}$

1: Initialize a random Gaussian matrix $W \in R^{J\times\widehat{R}}$;
2: Calculate sample matrix $Y = (XX^T)^q XW \in R^{I\times\widehat{R}}$;
3: Compute the QR decomposition of the sample matrix $Y = QR$;
4: Calculate the small-size matrix $A = Q^T X \in R^{\widehat{R}\times J}$;
5: Compute the SVD of the small-size matrix $A = \widehat{U}SV^T$;
6: Calculate the orthogonal matrix $U = Q\widehat{U}$;
7: **return** orthogonal matrices $U \in R^{I\times\widehat{R}}$, diagonal matrix $S \in R^{\widehat{R}\times\widehat{R}}$ and $V \in R^{J\times\widehat{R}}$

The advantage of using the overestimated rank of the matrix is that it can achieve a more accurate approximation of the matrix. (Chen *et al.*) [129] improved the approximation of SVD decomposition by integrating multiple random sketches, that is, multiplying the input matrix X by a set of random Gaussian matrices. (Halko *et al.*) [96] used a special sampling matrix to greatly reduce the execution time of the algorithm while reducing complexity. However, for a matrix with a slow singular value decay, this method will result in a lower accuracy of SVD.

Many researchers developed a variety of different algorithms to solve the HOSVD decomposition. For details, please refer to (Vannieuwenhoven *et al.* [101]; Austin *et al.* [138]; Constantine *et al.* [103]).

Compared with the truncated SVD decomposition of the standard matrix, the tensor HOSVD decomposition does not produce the best multiple linear rank, but only the weak linear rank approximation (De Lathauwer *et al.*) [73]:

$$\|\underline{X} - \underline{A} \times_{1m} B_1 \times_{2m} B_2 \cdots \times_{Nm} B_N\| \leq \sqrt{N}\|\underline{X} - \widehat{\underline{X}}_{Prefect}\| \tag{47}$$

where $\widehat{\underline{X}}_{Prefect}$ is the best approximation for $\underline{X}$.

In order to find an accurate approximation of Tucker decomposition, researchers have extended the alternating least squares method to the higher-order orthogonal iterations (Jeon *et al.* [56]; Austin *et al.* [138];Constantine *et al.* [103]; De Lathauwer *et al.* [74]). For details, please refer to Algorithm 4.

Algorithm 4 The Higher-Order Orthogonal Iterations (Austin *et al.* [138]; De Lathauwer *et al.* [74])

Input:

The Nth-order input tensor $\underline{X} \in R^{I_1\times I_2\cdots I_N}$ decomposed by Tucker.

Output:

the core tensor $\underline{A}$ and factor orthogonal matrices B_n,$B_n^T B_n = I_{R_n}$

1: Initialize all parameters via the Truncated HOSVD by Algorithm 2;
2: **while** the cost function $\|\underline{X} - \underline{A} \times_{1m} B_1 \cdots \times_{Nm} B_N\|_F^2$ does not reach convergence **do**
3: **for** n=1 to N **do**
4: $\underline{Y} \leftarrow \underline{X} \times_{(p\neq n)m} (B_p^T)$;
5: $Z \leftarrow \underline{Y}_{mn}(\underline{Y}_{mn})^T \in R^{R\times R}$;
6: $B_n \leftarrow$ leading R_n eigenvectors of Z;
7: **end for**
8: $\underline{A} \leftarrow \underline{Y} \times_{Nm} (B_N^T)$;
9: **end while**
10: **return** the core tensor $\underline{A}$ and factor matrices B_n

When the size of the original tensor is too large(too many elements), it will result in insufficient memory, and finally the computational complexity may also increase. In this case, the operation can be simplified in the form of a matrix product. Simply put, the mode-n product of the tensor and the matrix is converted into the product of the general matrix to simplify the operation and reduce the memory (see figure 20).

For the large size tensor, another way to simplify the operation is to use the blocking method. It simply divides the original tensor and the factor matrix into blocks, and then performs the mode-n product between the small size matrix and the small size tensor (see figure 21).

As seen from the figure 21, we divide the input tensor $\underline{X}$ into small pieces $\underline{X}_{(x_1,x_2,\cdots,x_N)}$. Similarly, we divide the factor matrix B_n^T into $B_{(x_n,b_n)}$. The tensor $\underline{A}^n$ remained by the mode-n product of the matrix and the tensor is equal to:

$$\underline{A}^n_{(x_1,x_2,\cdots,b_n,\cdots,x_N)} = \sum_{x_n=1}^{X_n} \underline{X}_{(x_1,x_2,\cdots,b_n,\cdots,x_N)} \times_{nm} (B_n(x_n, b_n))^T \tag{48}$$

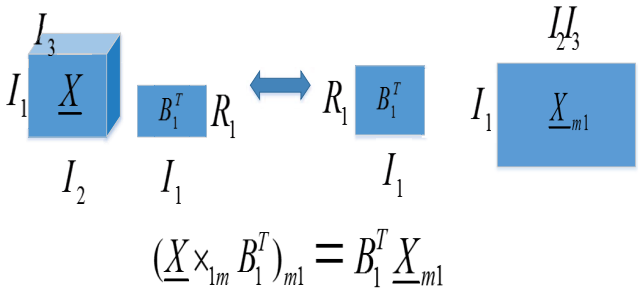

FIGURE 20. The mode-n product of a factor matrix and a large-size 3rd-order tensor is shown in the figure. When the tensor size is very large, the mode-n matricization of the tensor can be first performed, and then multiplied by the factor matrix to simplify the calculation.

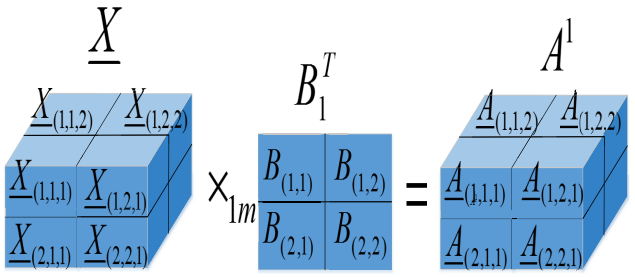

FIGURE 21. The mode-n product of a factor matrix and a large-size 3rd-order tensor is shown in the figure. When the tensor/matrix size is very large, we can also divide the large-size tensor/matrix into several small-size tensors/matrices, and then operate on the small-size tensors/matrices after blocking, thus simplifying the operation [52].

5) THE TENSOR-SVD(T-SVD) DECOMPOSITION

Similar to the SVD decomposition of a matrix, when we extend the matrix to a 3rd-order tensor, we call it the Tensor-SVD decomposition.

$$\underline{X} = \underline{U} \times_{(2,1)(3,3)} \underline{S} \times_{2,1} \underline{V}^* \tag{49}$$

where $\underline{X} \in R^{I_1 \times I_2 \times I_3}$, $\underline{U} \in R^{I_1 \times I_1 \times I_3}$, $\underline{V} \in R^{I_2 \times I_2 \times I_3}$. $\underline{S} \in R^{I_1 \times I_2 \times I_3}$ means f- diagonal tensor, each of its frontal slices is a diagonal matrix. We draw a figure to show the t-svd decomposition (see figure 22).

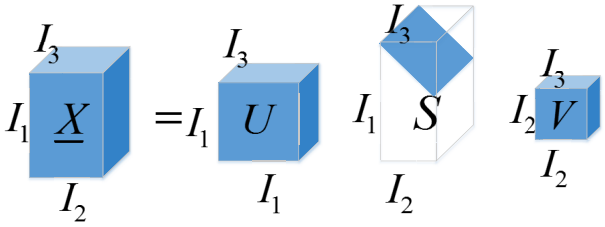

FIGURE 22. T-SVD of a 3rd-order tensor [20].

6) THE TENSOR CROSS-APPROXIMATION

Before we discuss the concept of the Tensor Cross-Approximation, we first introduce some concepts of matrix cross approximation. (Bebendorf *et al.* [85]; Khoromskij and Veit [16]) proposed the concept of the matrix cross approximation(MCA). The main role of the MCA is to reduce the size of the original large-size matrix by finding a linear combination of several components of the matrix, thereby decreasing computational complexity and computational memory. These components are usually a small fraction of the original matrix. This method has a premise that the original matrix is highly redundant, so it can be approximated by a small size matrix with some marginal information lost.

We illustrate the MCA in figure 23.

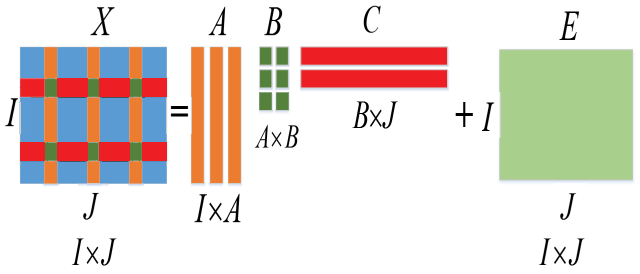

FIGURE 23. Schematic diagram of MCA, $X = ABC + E$, $A \in R^{I \times A}$, $B \in R^{A \times B}$, $C \in R^{B \times J}$, $E \in R^{I \times J}$ [3].

From figure 23 we give the specific formula of the MCA method:

$$X = ABC + E \tag{50}$$

where $A \in R^{I \times A}$ is a small size matrix obtained by selecting appropriate A columns from the original matrix X. $B \in R^{A \times B}$ is a small size matrix obtained by selecting the appropriate B rows from the original matrix X. $C \in R^{B \times J}$ is a small size matrix obtained from the appropriate selected B rows in the original matrix X. $E \in R^{I \times J}$ is the redundant matrix (error matrix).

Obviously, if the elements of the error matrix are small enough, we can convert the MCA decomposition of X above into a CR matrix decomposition.

$$X \approx CR \tag{51}$$

where $C = A \in R^{I \times A}$, $R = BC \in R^{A \times J}$ or $C = AB \in R^{I \times B}$, $R = C \in R^{B \times J}$.

Note that in order to reduce the size, $A \ll J$ and $B \ll I$, and minimize the F norm of the redundancy matrix $\|E\|_F$, the choices of A and B are also very important. Generally, if A and B are given, then the three matrices can be obtained according to the method as shown in figure 23(split the original matrix and then get three matrices from it). Another special property is that when $rank(X) \leq min(A, B)$, the matrix cross-approximation solution is exact or the error matrix E at this time is very small and can be ignored, i.e., $X = ABC$.

Now we extend the concept of the MCA to the form of tensor, i.e., tensor cross-approximation (TCA). There are usually two ways to implement TCA.

1. (Mahoney *et al.*) [93] extended MCA to the matrix form of tensor data (that is, find the matricization of the tensors and then implement MCA).

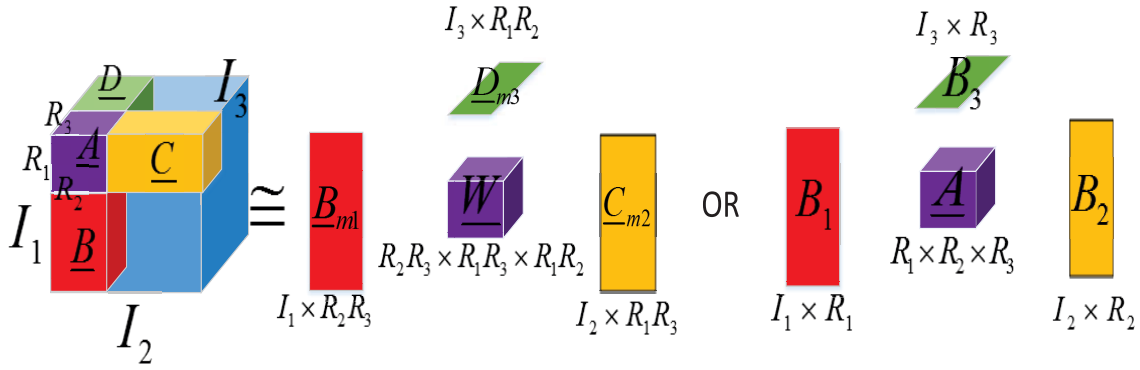

FIGURE 24. The schematic diagram of TCA is similar to MCA. It is noted that R_1, R_2 and R_3 are selected appropriately, and then four new tensors $\underline{A}$, $\underline{B}$, $\underline{C}$, $\underline{D}$ are formed. The rightmost is the equivalent Tucker decomposition diagram. For detailed derivation, please refer to formula 52.

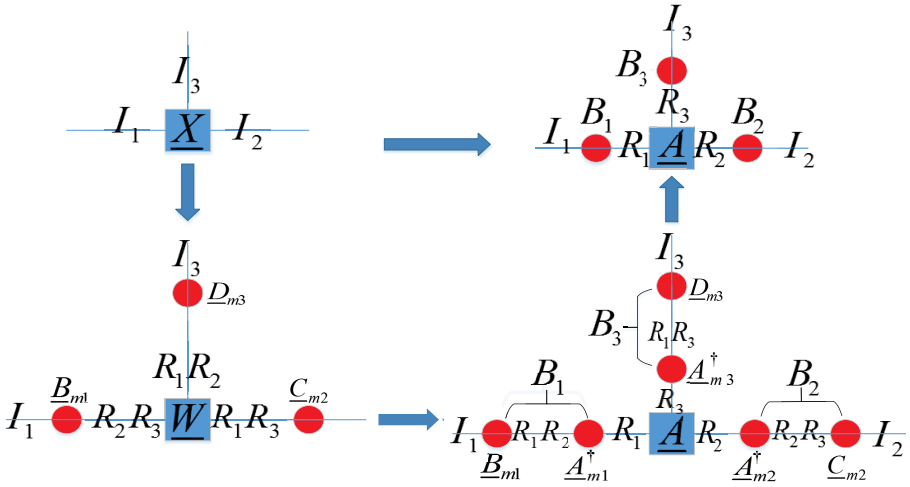

FIGURE 25. A simple tensor network diagram of TCA and Tucker decomposition. Here is a schematic diagram of the conversion of TCA and Tucker decomposition [3].

2. (Caiafa and Cichocki) [17] proposed a Fiber Sampling Tucker Decomposition that operates directly on the input matrix, but with the premise that it is based on low rank Tucker decomposition. Since tensor usually has a good low-rank Tucker decomposition, FSTD algorithm is often used. Figure 24 and 25 shows the TCA by FSTD algorithm.

We can see from figure 24 and 25 that the FSTD algorithm first finds a suitable cross tensor from the original input tensor, and then changes the size of the core tensor. Specifically by the formula:

$$\begin{aligned}\underline{X} &= \underline{A} \times_{1m} B_1 \times_{2m} B_2 \times_{3m} B_3 \\ &= \underline{W} \times_{1m} \underline{B}_{m1} \times_{2m} \underline{C}_{m2} \times_{3m} \underline{D}_{m3}\end{aligned} \tag{52}$$

where the first equation is the standard Tucker decomposition. In the second equation, where $\underline{B}_{m1} \in R^{I_1 \times R_2 R_3}$, $\underline{C}_{m2} \in R^{I_2 \times R_1 R_3}$, $\underline{D}_{m3} \in R^{I_3 \times R_1 R_2}$, $\underline{W} = \underline{A} \times_{1m} \underline{A}^{\dagger}_{m1} \times_{2m} \underline{A}^{\dagger}_{m2} \times_{3m} \underline{A}^{\dagger}_{m3} \in R^{R_2 R_3 \times R_1 R_3 \times R_1 R_2}$. Note that $\underline{B}_{m1}\underline{A}^{\dagger}_{m1} = B_1$, $\underline{C}_{m2}\underline{A}^{\dagger}_{m2} = B_2$, $\underline{D}_{m3}\underline{A}^{\dagger}_{m3} = B_3$. The above is for the 3rd-order tensor, and when the dimension becomes 2(the matrix), it is easy to see that the TCA degenerates into MCA.

For an Nth-order tensor, the formula for FSTD is as follows (Caiafa and Cichocki, 2015) [17]:

$$\begin{aligned}\underline{X} &= \underline{A} \times_{1m} B_1 \times_{2m} B_2 \times \cdots \times_{Nm} B_N \\ &= \underline{W} \times_{1m} \underline{C}^1_{m1} \times_{2m} \underline{C}^2_{m2} \cdots \times_{Nm} \underline{C}^N_{mN}\end{aligned} \tag{53}$$

For a 3rd-order tensor, the four cross tensors of the above FSTD($\underline{W}$, $\underline{B}$, $\underline{C}$, $\underline{D}$) can be obtained by random projection (see formula 51), as follows:

$$\begin{aligned}\underline{W} &= \underline{X} \times_{1m} B_1 \times_{2m} B_2 \times_{3m} B_3 \in R^{R_1 \times R_2 \times R_3} \\ \underline{B} &= \underline{X} \times_{2m} B_2 \times_{3m} B_3 \in R^{I_1 \times R_2 \times R_3} \\ \underline{C} &= \underline{X} \times_{1m} B_1 \times_{3m} B_3 \in R^{R_1 \times I_2 \times R_3} \\ \underline{D} &= \underline{X} \times_{1m} B_1 \times_{2m} B_2 \in R^{R_1 \times R_2 \times I_3}\end{aligned} \tag{54}$$

where $B_n \in R^{R_n \times I_n}$ are the projection matrices.

7) THE TENSOR TRAIN AND TENSOR CHAIN DECOMPOSITION

CP decomposition is a special case of Tucker decomposition. The core tensor of Tucker decomposition is further decomposed into hierarchical tree structure and becomes

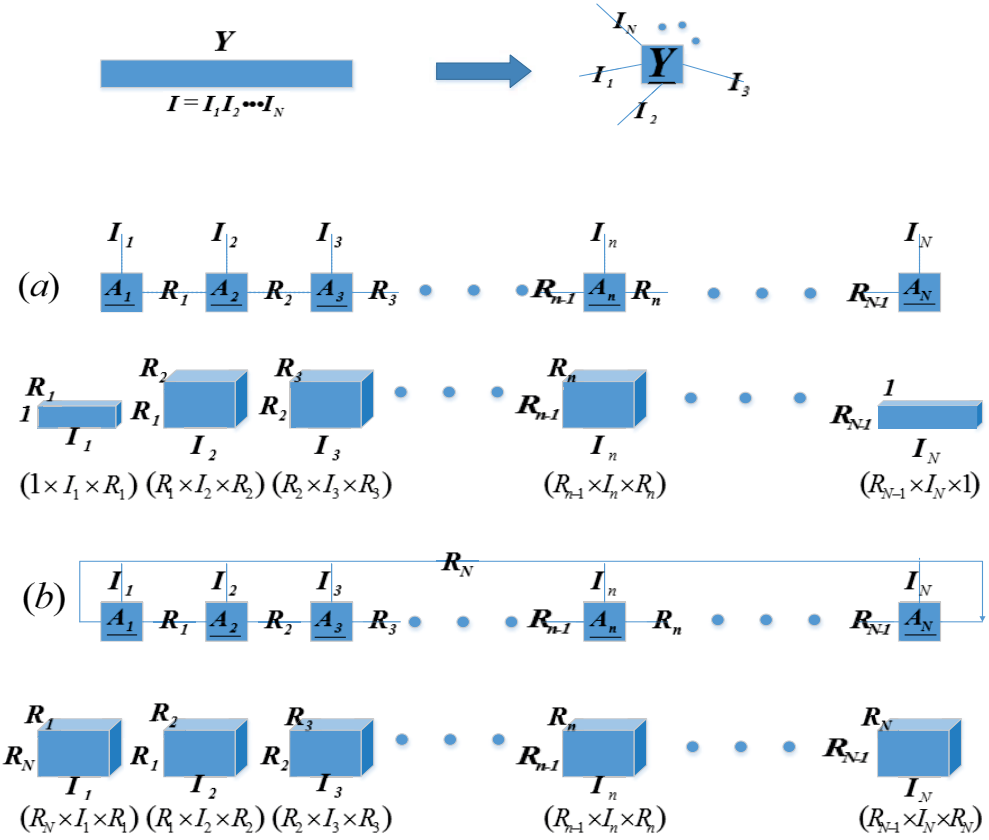

FIGURE 26. The TT and TC decomposition for a large size vector. Figure (a) first reorganizes the vector into a suitable Nth-order tensor, $\underline{Y} \in R^{I_1 \times I_2 \cdots \times I_N} \leftarrow y \in R^I, I = I_1 I_2 \cdots I_N$, and then TT and TC decomposition are performed on the Nth-order tensor. Figure (a) is TT decomposition, and Figure (b) is TC decomposition. Please refer to formula 55 for the TT decomposition of Nth-order tensor.

HT decomposition. The Tensor Chain(TC) decomposition is a special case of HT decomposition. The core tensor is in series and aligned, i.e., every core tensor has the same dimension, and at the same time, all the factor matrices are unit matrices. The advantage of having the same form of core tensor and unit matrix is that it can significantly reduce the amount of computation, facilitate subsequent optimization, and so on. The Tensor Train(TT) decomposition is also a special case of HT decomposition. (Oseledet [60] and Oseledet and Tyrtyshnikov [61]) first put forward the concept of TT decomposition. The only difference between TT decomposition and TC decomposition is that the dimension of the first and the Nth core tensor is one less than the dimension of the intermediate N-2 core tensors in TT decomposition. In different domains, TT decomposition has different names. Generally speaking, in the field of physics, when we refer to the Tensor Chain(TC) decomposition as the Matrix Product State (MPS) decomposition with periodic boundary conditions(PBC), we also refer to the TT decomposition as the Matrix Product State (MPS) decomposition with the Open Boundary Conditions. Before we give the concrete expression, we draw a picture to give an intuitive explanation of the TT decomposition and the TC decomposition (see figure 26 and figure 27).

In figure 26 and figure 27, we first transform the large size vector and matrix into the Nth-order and 2Nth-order small size tensor, respectively. Then we decompose them by TT or TC. We can see that the only difference between TT decomposition and TC decomposition is that TC decomposition connects the first core tensor and the last core tensor with a single line R_N.

Then we give a concrete mathematical expression of TT decomposition of an Nth-order tensor $\underline{Y} \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$.

$$\underline{Y} = \underline{A}_1 \times_{3,1} \underline{A}_2 \cdots \times_{3,1} \underline{A}_N \tag{55}$$

where $\underline{A}_n \in R^{R_{n-1} \times I_n \times R_n}$, $R_0 = R_N = 0, n = 1, 2, \cdots, N$

$$y_{i_1, i_2, \cdots, i_N} = \sum_{r_1, r_2, \cdots, r_{N-1}=1}^{R_1, R_2, \cdots, R_{N-1}} a^1_{1, i_1, r_1} a^2_{r_1, i_2, r_2} \cdots a^{N-1}_{r_{N-2}, i_{N-1}, r_{N-1}} a^N_{r_{N-1}, i_N, 1} \tag{56}$$

where $y_{i_1, i_2, \cdots, i_N}$ and $a^n_{r_{n-1}, i_n, r_n}$ are entries of $\underline{Y}$ and $\underline{A}_n$, respectively.

$$\underline{Y} = \sum_{r_1, r_2, \cdots, r_{N-1}=1}^{R_1, R_2, \cdots, R_{N-1}} a_1^{1, r_1} \circ a_2^{r_1, r_2} \circ \cdots \circ a_{N-1}^{r_{N-2}, r_{N-1}} a_N^{r_{N-1}, 1} \tag{57}$$

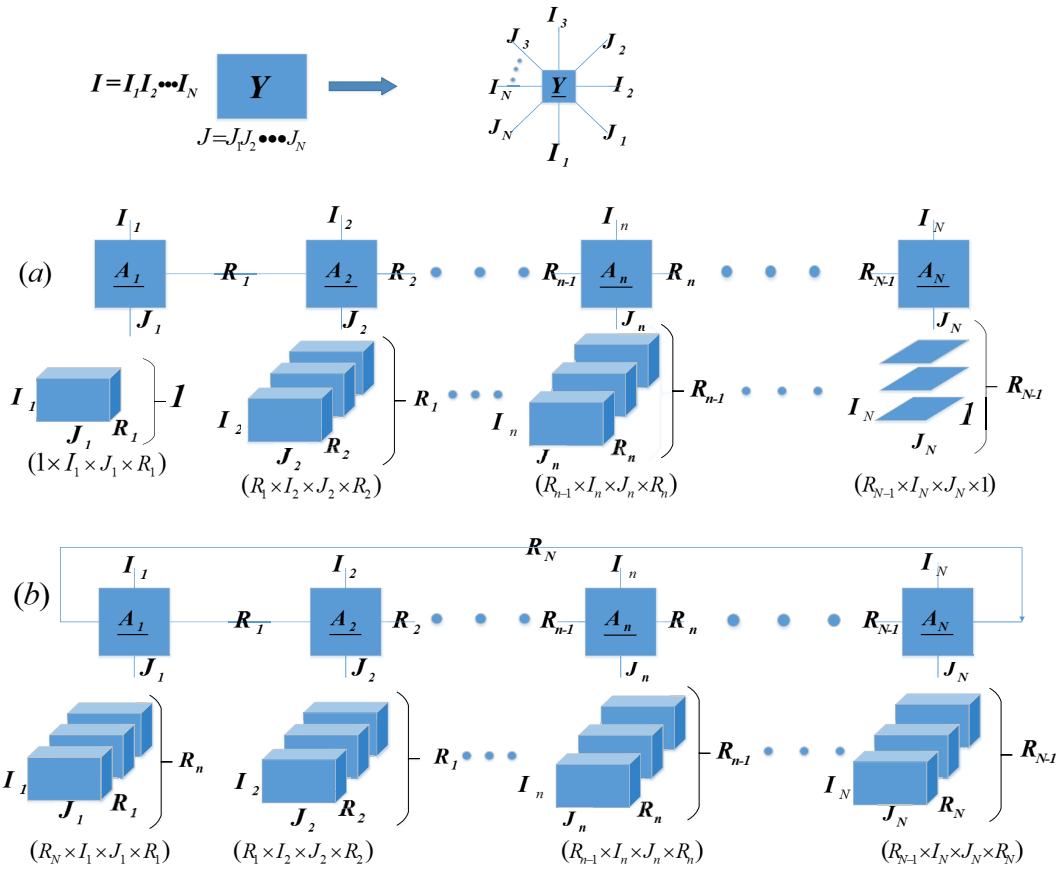

FIGURE 27. The TT and TC decomposition for a large size matrix. Figure (a) first reorganizes the matrix into a suitable 2Nth-order tensor, $\underline{Y} \in R^{I_1 \times J_1 \cdots \times I_N \times J_N} \leftarrow Y \in R^{I \times J}, I = I_1 I_2 \cdots I_N, J = J_1 J_2 \cdots I_N$, and then TT and TC decomposition are performed on the 2Nth-order tensor. Figure (a) is TT decomposition, and Figure (b) is TC decomposition. Please refer to formula 58 for the TT decomposition of 2Nth-order tensor.

where $a_n^{r_{n-1},r_n} = \underline{A}_n(r_{n-1},:,r_n) \in R^{I_n}$ are tensor fiber(vectors).

The above three formulas are TT decomposition formula corresponding to the large-size vector decomposed into Nth-order tensors (that is, figure 26). Similar to the TT decomposition for the Nth-order tensor, **the TT decomposition for the 2Nth-order tensor (see figure 27) is as follows**:

$$\underline{Y} = \underline{A}_1 \times_{4,1} \underline{A}_2 \cdots \times_{4,1} \underline{A}_N \tag{58}$$

where $\underline{A}_n \in R^{R_{n-1} \times I_n \times J_n \times R_n}, R_0 = R_N = 0, n = 1, 2, \cdots, N$

$$y_{i_1,i_2,\cdots,i_N} = \sum_{r_1,r_2,\cdots,r_{N-1}=1}^{R_1,R_2,\cdots,R_{N-1}} a^1_{1,i_1,j_1,r_1} a^2_{r_1,i_2,j_2,r_2} \cdots a^{N-1}_{r_{N-2},i_{N-1},j_{N-1},r_{N-1}} a^N_{r_{N-1},i_N,j_N,1} \tag{59}$$

where $y_{i_1,i_2,\cdots,i_N}$ and $a^n_{r_{n-1},i_n,j_n,r_n}$ are entries of $\underline{Y}$ and $\underline{A}_n$, respectively.

$$\underline{Y} = \sum_{r_1,r_2,\cdots,r_{N-1}=1}^{R_1,R_2,\cdots,R_{N-1}} A_1^{1,r_1} \circ A_2^{r_1,r_2} \circ \cdots \circ A_{N-1}^{r_{N-2},r_{N-1}} A_N^{r_{N-1},1} \tag{60}$$

where $A_n^{r_{n-1},r_n} = \underline{A}_n(r_{n-1},:,:,r_n) \in R^{I_n \times J_n}$ are tensor slice(matrices).

Similarly, the 3rd-order large-size tensor or higher-order large-size tensor can be decomposed by TT in a similar way (by decomposing them into 3Nth-order or higher tensor.)

Here we no longer give the mathematical expression of the TC decomposition, because there is almost no difference between the TT decomposition and TC decomposition (mainly the first and last core tensors have a dimension with a size of R_n).

Here we give three common methods. The first is the product form between the core tensor contractions, the second is the expression between the scalars and the third is the outer product of tensor slice or the outer product of tensor fiber. There are some other mathematical expressions for other uses, such as, the TT decomposition can be calculated by performing the mode-n matricization of the core tensor and then we can use the strong Kronecker product or tensor slices to calculate. Those who are interested can refer to (Cichocki *et al.*) [3].

Similar to the CP rank,**we define the TT rank.**

$$r_{TT}(\underline{Y}) = (R_1, R_2, \cdots, R_{N-1}),$$
$$R_n = rank(\underline{Y}_{mcn}) = r(\underline{Y}_{mcn}) \tag{61}$$

Here we add a concept. We have previously introduced the definition of mode-n matricization of tensor. But in fact, there are two ways to perform the matricization of tensor. One of them is to extract one dimension as the first dimension of the resulting matrix, and the remaining N-1 dimensions as the second dimension. The other is to extract n dimensions from the original tensor as the first dimension of the resulting matrix, and the remaining (N-n) dimensions as the second dimension. **We call the latter mode-n cannonical matricization of the tensor**:

$$mat(\underline{Y})_{cn} = \underline{Y}_{mcn} \in R^{I_1 \cdots I_n \times I_{n+1} \cdots I_N} \tag{62}$$

where m means matricization, c means cannonical, n means mode-n.

According to the mathematical expression of TT decomposition and the definition of TT rank, we give the computational complexity of TT decomposition.

$$\sum_{n=1}^{N} R_{n-1} I_n R_n \sim O(NIR^2), R = \max_n R_n, I = \max_n I_n \tag{63}$$

We can see from the formula that the complexity is related to the TT rank. Thus, we need to find a suitable low rank TT decomposition to reduce the complexity.

8) THE TENSOR NETWORKS(DECOMPOSITIONS) WITH CYCLES

In the above sections, we briefly introduced TT decomposition, HT decomposition and other tree tensor networks. We should note that all the tensor decomposition networks mentioned above do not contain a circle(except TC). We also mentioned in the previous section that the TT rank usually increases with the growth of the dimension of original data tensor that needs to be decomposed. As the depth of decomposition increases for an arbitrary tree-shaped tensor network, the TT rank will also increase. In order to reduce the TT rank, researchers invented some layered tensor networks with Loops. (Verstraete *et al.* [32]; Schuch *et al.* [100]) proposed Projected Entangled Pair States(PEPS) and Projected Entangled Pair Operators(PEPO), respectively (see figure 28). In these two kinds of tensor networks, they replaced the 3rd-order core tensors of the original TT decomposition with 5th and 6th-order core tensors, respectively. But they reduced tensor rank at the expense of higher complexity because the original 3rd-order core tensor rises to 5th order, 6th order.

Sometimes for some higher order tensors in science and physics, it may not be enough to reduce the rank for the above two kinds of networks. Some researchers have proposed new tensor networks with more circles. (Giovannetti *et al.* [135]; Matsueda [50]) produced the Honey-Comb Lattice(HCL) and the Multi-scale Entanglement Renormalization Ansatz(MERA), respectively (see figure 29). They used the 3rd and 4th-order core tensors, respectively. However, as the number of cycles increases, the overall computational complexity of the network increases, i.e., we need to calculate more circles. In short, in order to reach the balance between rank and complexity in practice, the network will be selected according to the need.

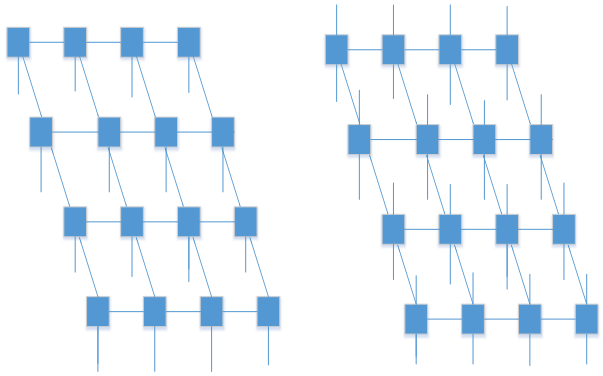

FIGURE 28. Projected Entangled Pair States(PEPS) and Projected Entangled Pair Operators(PEPO). PEPS on the left and PEPO on the right. The blue rectangles represent core tensors. They use the 5th and 6th-order core tensors, respectively [3].

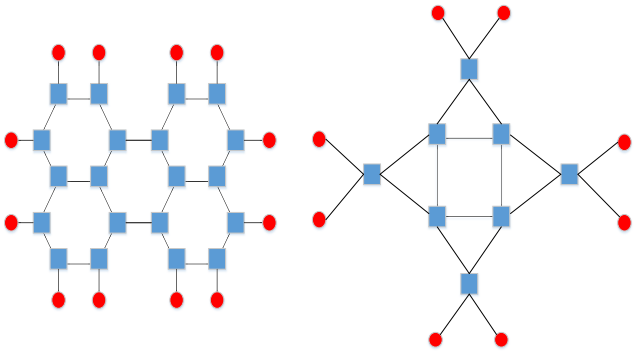

FIGURE 29. Honey-Comb Lattice(HCL) and the Multi-scale Entanglement Renormalization Ansatz(MERA). HCL on the left and MERA on the right. The blue rectangle represents core tensors and the red circle represents factor matrices. HCL uses the 3rd-order core tensors while MERA uses the 3rd-order and 4th-order tensors.

Compared with the former two tensor networks with cycles, the size and dimension of the core tensor in MERA are usually smaller, so the number of unknown parameters (variables or free parameters) will be reduced, and the corresponding computational complexity will also be decreased. At the same time, the MERA network with cycles can help us find the relationship and interaction between tensor and free parameters. In general, the main idea of these four methods is to reduce TT rank by increasing the number of core tensors and reducing the size of core tensors but usually at the cost of increasing computational complexity. The advantage of small size tensor is that it is easier to manage, and it can reduce the number of free parameters in the network. For a single small-scale tensor, the calculation is relatively simple. At the same time, we can see that due to the cycle structure, these four networks can usually describe the correlation between variables well.

D. THE NATURE AND ALGORITHM OF TT DECOMPOSITION

1) BASIC OPERATIONS IN TT DECOMPOSITION

If large-size tensors are given in the form of TT decomposition, then many calculations can be performed on the small-size core tensors. By performing operations on small-size core tensors, the unknown parameters can be reduced effectively, and the operations can be simplified to achieve the effect of the optimization algorithm.

Consider two Nth-order tensors in TT decomposition:

$$\begin{aligned}\underline{X} &= \underline{X}_1 \times_{3,1} \underline{X}_2 \cdots \times_{3,1} \underline{X}_N \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N} \\ \underline{Y} &= \underline{Y}_1 \times_{3,1} \underline{Y}_2 \cdots \times_{3,1} \underline{Y}_N \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}\end{aligned} \tag{64}$$

where the core tensors $\underline{X}_n \in R^{R_{n-1} \times I_n \times R_n}$, $\underline{Y}_n \in R^{Q_{n-1} \times I_n \times Q_n}$ and their TT ranks are $r_{TT}(\underline{X}) = (R_1, \cdots, R_{N-1})$ and $r_{TT}(\underline{Y}) = (Q_1, \cdots, Q_{N-1})$, respectively. Note that the size and dimension of two tensors are the same. **Their operations have the following properties**:

1. the Hadamard product of two tensors:

$$\underline{Z} = \underline{X} \circledast \underline{Y} = \underline{Z}_1 \times_{3,1} \underline{Z}_2 \cdots \times_{3,1} \underline{Z}_N \tag{65}$$

We can use the tensor slice to represent the core tensor Z.

$$\underline{Z}_n^{(i_n)} = \underline{X}_n^{(i_n)} \otimes_L \underline{Y}_n^{(i_n)}, \quad n = 1, \cdots, N, i_n = 1, \cdots, I_n \tag{66}$$

where $\underline{Z}_n \in R^{R_{n-1}Q_{n-1} \times I_n \times R_n Q_n}$ is the core tensor and $\underline{Z}_n^{(i_n)} \in R^{R_{n-1}Q_{n-1} \times R_n Q_n}$, $\underline{X}_n^{(i_n)} \in R^{R_{n-1} \times R_n}$, $\underline{Y}_n^{(i_n)} \in R^{Q_{n-1} \times Q_n}$ is the tensor slice (fix the second dimension i_n to get).

2. the sum of two tensors:

$$\underline{Z} = \underline{X} + \underline{Y} \tag{67}$$

where its TT rank $r_{TT}(\underline{Z}) = r_{TT}(\underline{X}) + r_{TT}(\underline{Y}) = (R_1 + Q_1, R_2 + Q_2, \cdots, R_N + Q_N)$, similar to the previous one, we can still use tensor slice to represent Z.

$$\underline{Z}_n^{(i_n)} = \begin{bmatrix} \underline{X}_n^{(i_n)} & 0 \\ 0 & \underline{Y}_n^{(i_n)} \end{bmatrix}, \quad n = 2, 3, 4, \cdots, N-1 \tag{68}$$

Note that the tensor slices of the first and last core tensors are as follows:

$$\underline{Z}_n^{(1)} = \begin{bmatrix} \underline{X}_n^{(1)} & \underline{Y}_n^{(1)} \end{bmatrix}, \quad \underline{Z}_n^{(i_N)} = \begin{bmatrix} \underline{X}_n^{(i_N)} \\ \underline{Y}_n^{(i_N)} \end{bmatrix} \tag{69}$$

3. the quantitative product of two tensors:

$$\begin{aligned} A_N &= \underline{X} \bullet \underline{Y} \\ A_n &= \underline{X}_n \times_{1,2} (\underline{Y}_n \times_{1m} A_{n-1}) \in R^{R_n \times Q_n}, \quad n = 1, \cdots, N \end{aligned} \tag{70}$$

Then we calculate the final result by iterative method, the specific process reference algorithm 5.

4. the multiplication of large-size matrix and vector:

$$A\boldsymbol{x} \approx \boldsymbol{y} \tag{71}$$

where $A \in R^{I \times J}$, $\underline{X} \in R^{J_1 \times J_2 \cdots \times J_N} \leftarrow \boldsymbol{x} \in R^J$, $J = J_1 J_2 \cdots J_N$, $\underline{Y} \in R^{I_1 \times I_2 \cdots \times I_N} \leftarrow \boldsymbol{y} \in R^I$, $I = I_1 I_2 \cdots I_N$

Algorithm 5 The Quantitative Product of Two Tensors Expressed in the Form of TT Decomposition

Input:

The two Nth-order tensors $\underline{X} = \underline{X}_1 \times_{3,1} \underline{X}_2 \cdots \times_{3,1} \underline{X}_N \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, $\underline{Y} = \underline{Y}_1 \times_{3,1} \underline{Y}_2 \cdots \times_{3,1} \underline{Y}_N \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, where $\underline{X}_n \in R^{R_{n-1} \times I_n \times R_n}$, $\underline{Y}_n \in R^{Q_{n-1} \times I_n \times Q_n}$, $R_0 = Q_0 = R_N = Q_N = 1$.

Output:

the quantitative product of the two tensors

Initialize $A_0 = 1$;

for n=1 to N **do**

$(\underline{Z}_n)_{m1} = A_{n-1}(\underline{Y}_n)_{m1} \in R^{Q_{n-1} \times Q_n}$;

$A_n = ((\underline{X}_n)_{mc2})^T (\underline{Z}_n)_{mc2} \in R^{R_n \times Q_n}$;

end for

return $A_N = \underline{X} \bullet \underline{Y} \in R$

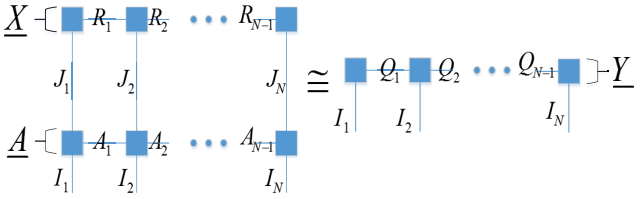

FIGURE 30. The multiplication of large-size matrix and vector. $A\boldsymbol{x} \approx \boldsymbol{y}, A \in R^{I \times J}, \underline{X} \in R^{J_1 \times J_2 \cdots \times J_N} \leftarrow x \in R^J, J = J_1 J_2 \cdots J_N, \underline{Y} \in R^{I_1 \times I_2 \cdots \times I_N} \leftarrow y \in R^I, I = I_1 I_2 \cdots I_N$ [3].

are decomposed in TT. We give an intuitive picture to show it(see figure 30).

As we can see from the figure 30, $\underline{A}_n \in R^{A_{n-1} \times I_n \times J_n \times A_n}$, $\underline{X}_n \in R^{R_{n-1} \times J_n \times R_n}$, $\underline{Y}_n \in R^{Q_{n-1} \times I_n \times Q_n}$. If starting from the form of the outer product of the TT decomposition, it is as follows:

$$\begin{aligned} \underline{A} &= \sum_{a_1, a_2, \cdots, a_{N-1}=1}^{A_1, A_2, \cdots, A_{N-1}} A_1^{1, a_1} \circ A_2^{a_1, a_2} \circ \cdots \circ A_{N-1}^{a_{N-2}, a_{N-1}} A_N^{a_{N-1}, 1} \\ \underline{X} &= \sum_{r_1, r_2, \cdots, r_{N-1}=1}^{R_1, R_2, \cdots, R_{N-1}} x_1^{1, r_1} \circ x_2^{r_1, r_2} \circ \cdots \circ x_{N-1}^{r_{N-2}, r_{N-1}} x_N^{r_{N-1}, 1} \\ \underline{Y} &= \sum_{r_1, r_2, \cdots, r_{N-1}=1}^{R_1, R_2, \cdots, R_{N-1}} y_1^{1, r_1} \circ y_2^{r_1, r_2} \circ \cdots \circ y_{N-1}^{r_{N-2}, r_{N-1}} y_N^{r_{N-1}, 1} \end{aligned} \tag{72}$$

then the multiplication of a matrix and a vector is equivalent to:

$$y_n^{r_{n-1}, r_n} = A_n^{a_{n-1}, a_n} x_n^{r_{n-1}, r_n}, Q_n = A_n R_n, \quad n = 1, 2, \cdots, N \tag{73}$$

Similarly, we can use the tensor network of TT decomposition to represent some loss functions (see figure 31).

Similar to the multiplication of matrices and vectors, TT decomposition can also be used to simplify the solution for multiplication between large-scale matrices and matrices, and here we omit its solution. Since the outer product

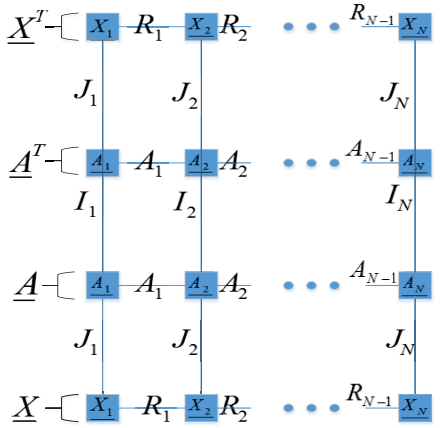

FIGURE 31. Special loss function represented by TT decomposition network. $J(x) = x^T A^T A x$, where x^T, A^T, A, x are all represented by TT decomposition.

calculation is relatively simple, we use the outer product expression of the TT decomposition to simplify the multiplication between the matrix and the vector. Of course, we can also use the TT decomposition expressed in the form of Kronecker or tensor contraction for simplified solution. For more calculations on TT decomposition, please refer to (Kazeev *et al.* [132]; Lee and Cichocki [99])

2) TT DECOMPOSITION SOLUTION

The solution of TT decomposition is similar to the solution of the truncated HOSVD algorithm mentioned above (see algorithm 2), and the following constraints need to be met:

$$(\|\underline{Y} - \widehat{\underline{Y}}\|_{l_2})^2 \leq \sum_{n=1}^{N-1} \sum_{k=R_n+1}^{I_n} (\sigma_k(\underline{Y}_{mcn}))^2 \tag{74}$$

where $\widehat{\underline{Y}}$ is the approximate estimated tensor of the original tensor. The l_2 norm of the tensor is equal to the Frobenius norm of the tensor. $\sigma_k(\underline{Y}_{mcn})$ represents the kth maximum singular value of the nth cannonical matricization of the input tensor $\underline{Y}$.

Under the above constraints, there are usually three basic ways to obtain the solution of TT decomposition.

1 **SVD-based TT algorithm (TT-SVD)**

2 **Algorithm based on low rank matrix decomposition(LRMD)**

3 **Restricted Tucker-1 decomposition(RT1D)**

SVD-based TT algorithm first performs mode-n matricization on the input tensor and then performs HOSVD decomposition (see figure 32 and algorithm 6).

Algorithm based on low rank matrix decomposition(LRMD) is similar to SVD-based TT algorithm (see figure 33 and algorithm 7). The only difference we noticed is that after performing the mode-n matricization of the first step, the original complex SVD decomposition operation is simplified by using matrix cross-approximation or CR decomposition.

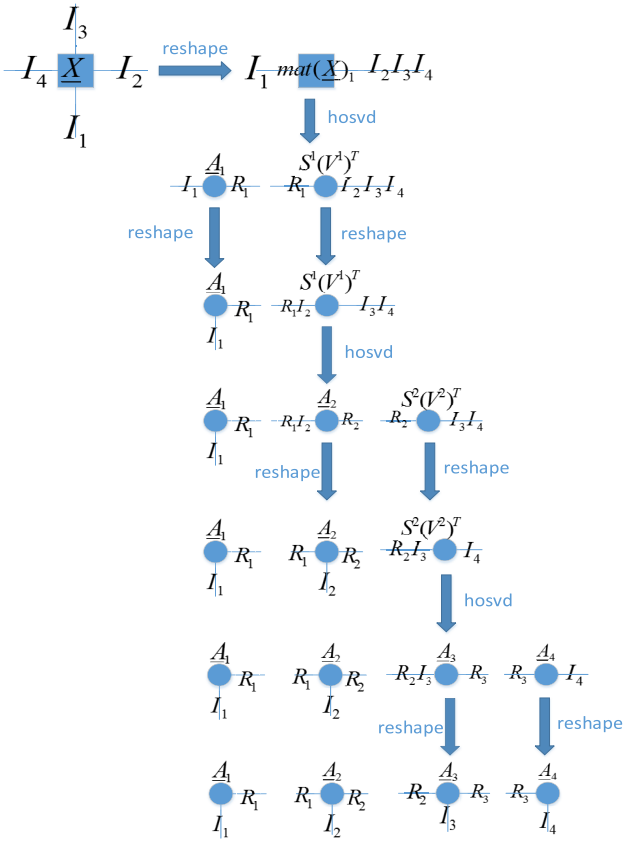

FIGURE 32. SVD-based TT algorithm (TT-SVD) [40] for a 4th-order tensor $\underline{X} \in R^{I_1 \times I_2 \times I_3 \times I_4}$. First,we perform the mode-n matricization of the tensor $\underline{X}$, here we perform the mode-1 matricization for convenience. Then we perform the SVD decomposition and execute algorithm 6 step by step.

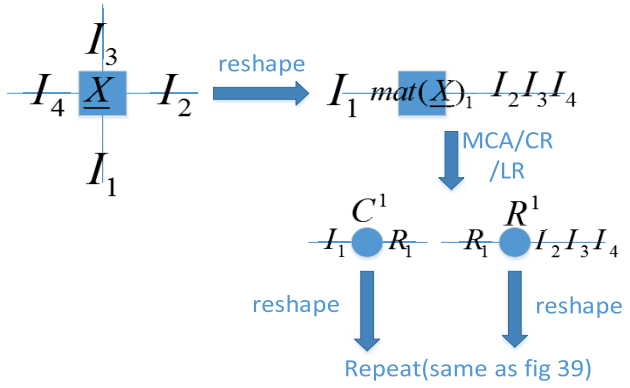

FIGURE 33. Algorithm based on low rank matrix decomposition for a 4th-order tensor $\underline{X} \in R^{I_1 \times I_2 \times I_3 \times I_4}$. First,we perform the mode-n matricization of the tensor $\underline{X}$, here we perform the mode-1 matricization for convenience. Then we perform the CR/MCA/LR or other low-rank matrix decomposition methods. Then step by step according to algorithm 7.

Note that in the above two methods, we constructed the mode-n matricization of a tensor and then performed matrix decomposition-related operations. The third method, Restricted Tucker-1 decomposition (RT1D), converts the original input tensor into 3rd-order tensor, and then performs Tucker-1 and Tucker-2 decomposition (see figure 34).

Algorithm 6 SVD-Based TT Algorithm (TT-SVD) [40]

Input:
The Nth-order tensor $\underline{X} \in R^{I_1 \times I_2 \times \cdots I_N}$ and accuracy ε

Output:
Approximate tensor of TT decomposition $\widehat{\underline{X}}$, such that $\|\underline{X} - \widehat{\underline{X}}\|_F \leq \varepsilon$

Initialize $R_0 = 1, Z_1 = \underline{X}_{m1}$;
for n=1 to N-1 **do**
  $[U_n, S_n, V_n^T] = [U_n^1, U_n^2][S_n^1, 0][V_{n1}^T, V_{n2}^T] = truncated - svd(\underline{A}_{mn}, \frac{\varepsilon}{\sqrt{N}})$;
  $\underline{A}_n = U_n^1$;
  Reshape $\underline{A}_n$ in the manner described in figure 32, $\underline{A}_n = \underline{A}_n .reshape([R_{n-1}, I_n, R_n])$;
  $Z_{n+1} = S_n V_n^T .reshape([R_n I_{n+1}, I_{n+2} I_{n+3} \cdots I_N])$;
end for
Compute the last core $\underline{A}_N = \underline{Z}_N .reshape([R_{n-1}, I_N, 1])$;
return the core $\underline{A}_1, \underline{A}_2, \cdots, \underline{A}_N$

Algorithm 7 Algorithm Based on Low Rank Matrix Decomposition (LRMD) (Taking CR Decomposition as an Example) [29]

Input:
The Nth-order tensor $\underline{X} \in R^{I_1 \times I_2 \times \cdots I_N}$ and accuracy ε

Output:
Approximate tensor of TT decomposition $\widehat{\underline{X}}$, such that $\|\underline{X} - \widehat{\underline{X}}\|_F \leq \varepsilon$

Initialize $R_0 = 1, Z_1 = \underline{X}_{m1}$;
for n=1 to N-1 **do**
  $[\underline{C}^n, R^n] = CR - decomposition(Z_n, \varepsilon)$;
  Choose the suitable R_n;
  Reshape $\underline{C}^n$ in the manner described in figure 33, $\underline{A}_n = \underline{C}^n .reshape([R_{n-1}, I_n, R_n])$;
  $Z_{n+1} = R^n .reshape([R_n I_{n+1}, I_{n+2} I_{n+3} \cdots I_N])$;
end for
Compute the last core $\underline{A}_N = \underline{Z}_N .reshape([R_{n-1}, I_N, 1])$;
return the core $\underline{A}_1, \underline{A}_2, \cdots, \underline{A}_N$

It can be seen from figure 34 that we first compress the original tensor into a 3rd-order tensor. It should be noted that we use the first and Nth dimensions of the original Nth-order tensor as the first and third dimensions of the new 3rd-order tensor, respectively. The remaining N-2 dimensions are all multiplied as the second dimension of the new 3rd-order tensor. Specifically as shown in the following formula:

$$\underline{Y}_1 = \underline{X}.reshape([I_1, I_2 I_3 \cdots I_{N-1}, I_N]) \tag{75}$$

For the new third-order tensor, we first perform the Tucker-1 decomposition. Then, according to the parity of N, we perform the Tucker-2 or Tucker-1 decomposition, respectively. Specifically as shown in algorithm 8.

Algorithm 8 Restricted Tucker-1 Decomposition (RT1D) [10]

Input:
The Nth-order tensor $\underline{X} \in R^{I_1 \times I_2 \times \cdots I_N}$ and accuracy ε

Output:
Approximate tensor of TT decomposition $\widehat{\underline{X}}$, such that $\|\underline{X} - \widehat{\underline{X}}\|_F \leq \varepsilon$

Initialize $R_0 = R_N = 1$, $\underline{Y}_1 = Tensorization(\underline{X}) \in R^{I_1 \times I_2 I_3 \cdots I_{N-1} \times I_N}$;
if N is an odd number **then**
  for n=1 to $\frac{N-1}{2}$ **do**
    $[B_n, \underline{Y}_{n+1}, B_{N+1-n}] = Tucker - 1 - decomposition(\underline{Y}_n, \varepsilon)$;
    Estimate $R_n = \text{size}(\underline{B}_n, 2)$, $R_{N-n} = \text{size}(\underline{B}_{N+1-n}, 2)$;
    Reshape $\underline{Y}_n$ in the manner described in figure 34, $\underline{Y}_{n+1} = \underline{Y}_{n+1}$. reshape $([R_{n-1} I_n, I_{n+1} \cdots I_{N-n}, I_{N-n+1} R_{N-n+1}])$, $\underline{A}_n = B_n .reshape([R_{n-1}, I_n, R_n])$, $\underline{A}_{N+1-n} = B_{N+1-n} .reshape([R_{N-n}, I_{N+1-n}, R_{N+1-n}])$;
  end for
  Compute the last core $\underline{A}_{\frac{N+1}{2}} = \underline{Y}_{\frac{N+1}{2}} .reshape([R_{\frac{N-1}{2}}, I_{\frac{N+1}{2}}, R_{\frac{N+1}{2}}])$;
else {N is an even number}
  for $n = 1$ to $\frac{N-2}{2}$ **do**
    $[B_n, \underline{Y}_{n+1}, B_{N+1-n}] = Tucker - 1 - decomposition(\underline{Y}_n, \varepsilon)$;
    Estimate $R_n = \text{size}(\underline{B}_n, 2)$, $R_{N-n} = \text{size}(\underline{B}_{N+1-n}, 2)$;
    Reshape $\underline{Y}_n$ in the manner described in figure 34, $\underline{Y}_{n+1} = \underline{Y}_{n+1} .reshape([R_{n-1} I_n, I_{n+1} \cdots I_{N-1}, R_{N-1}])$, $\underline{A}_n = B_n .reshape([R_{n-1}, I_n, R_n])$, $\underline{A}_{N+1-n} = B_{N+1-n} .reshape([R_{N-n}, I_{N+1-n}, R_{N+1-n}])$;
  end for
  $[B_{\frac{N}{2}}, \underline{Y}_{\frac{N+2}{2}}] = Tucker - 2 - decomposition(\underline{Y}_{\frac{N}{2}}, \varepsilon)$;
  Reshape $\underline{Y}_n$ in the manner described in figure 34, $\underline{A}_n = B_{\frac{N}{2}} .reshape([R_{\frac{N-2}{2}}, I_{\frac{N}{2}}, R_{\frac{N}{2}}])$, $\underline{A}_{\frac{N+2}{2}} = \underline{Y}_{\frac{N+2}{2}} .reshape([R_{\frac{N}{2}}, I_{\frac{N+2}{2}}, R_{\frac{N+2}{2}}])$;
end if
return the core $\underline{A}_1, \underline{A}_2, \cdots, \underline{A}_N$

3) TT TRUNCATION

In the previous section, we discussed the problem of increased complexity due to increased TT rank. Therefore, if we still use the TT decomposition, we need to adopt some approximate decomposition algorithms to reduce the TT rank. (Oseledets) [60] proposed an algorithm called TT Truncation. The algorithm first inputs a tensor with large TT rank. The goal of this algorithm is to find an approximate solution whose rank is much smaller than the original input tensor. For TT Truncation, please refer to algorithm 9 and figure 35).

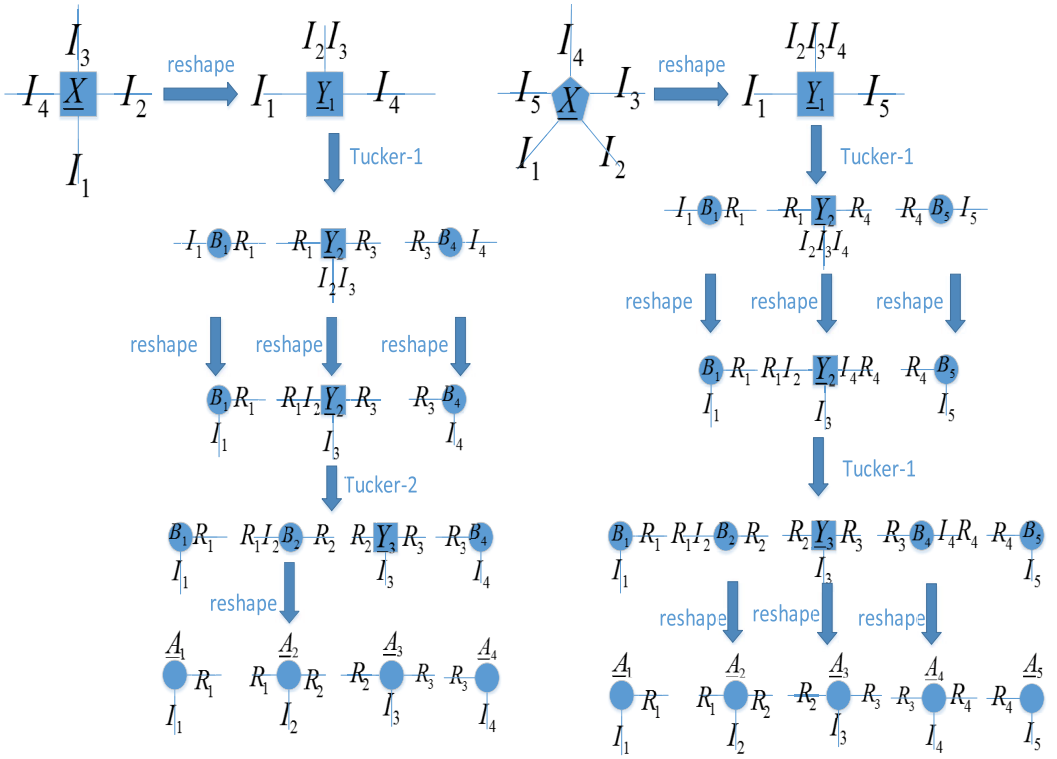

FIGURE 34. Restricted Tucker-1 decomposition(RT1D) [10] for a 4th-order tensor $\underline{X} \in R^{I_1 \times I_2 \times I_3 \times I_4}$ and a 5th-order tensor $\underline{X} \in R^{I_1 \times I_2 \times I_3 \times I_4 \times I_5}$. Similar to TT-SVD and LRMD, we first convert the original tensor into a new 3rd-order tensor, next perform the Tucker-1 decomposition, and then follow the algorithm 8 step by step. On the left is a schematic diagram of the 4th-order tensor and on the right is a schematic diagram of the 5th-order tensor.

It is noted that the algorithm 9 actually performs the Nth cannonical matricization of the core tensor and then performs a low rank matrix approximation (SVD and QR). We noticed that in the process of calculating the low rank matrix decomposition, the size of matrix will become smaller and smaller because of continuous iterative optimization, so the complexity will be continuously reduced in the process of performing decomposition. By TT Truncation, TT rank can be reduced to the utmost extent and the corresponding approximate tensor can be found, which greatly reduces the computational complexity and improves the efficiency for future data processing, mathematical operations, and so on. Of course, some researchers have developed a similar method for the HT decomposition. For details, please refer to (Kressner and Tobler) [25].

E. BRIEF SUMMARY FOR PART ONE

Part one mainly introduced the basic knowledge about tensor, including the definition of tensor, the operation of tensor, and the concept of tensor decomposition. As a new technique, tensor decomposition can reduce the computational complexity and memory by decomposing the tensor into lower-order tensors, matrices, and vectors. At the same time, it can preserve the data structure, effectively reduce the dimension, avoid the curse of dimension problems, and extract the important parts we need from the correlation. At the same time, the biggest feature of tensor decomposition is that the increase of dimension will lead to the non-uniqueness of decomposition. So we usually want to get an approximate solution of it instead of an exact solution, so that don't waste too much computation time and can get a good approximation of the original data.

Due to the limited space of this survey, there are some new tensor decompositions that are not covered in detail in this survey, such as t-svd(Zhang and Aeron) [165], tensor ring decomposition(Zhao *et al.*) [109]. The above introduction is several important tensor decompositions in this survey, and has important applications in part two. At the same time, some of these decomposition algorithms have their own advantages or limitations.

For CP decomposition, due to its particularity, if a certain constraint condition is imposed on the factor matrices or core tensor, an accurate solution can be obtained. The constraint is mainly determined according to the required environment. The advantage is that it can extract the structured information of the data, which helps better extract and process the required data, and improves the accuracy of the application in the future. For the Tucker decomposition, since the decomposition is general, the solution is usually more, so it is usually considered to impose a constraint term, such as the orthogonal constraint we mentioned above. Then the Tucker decomposition becomes HOSVD decomposition.

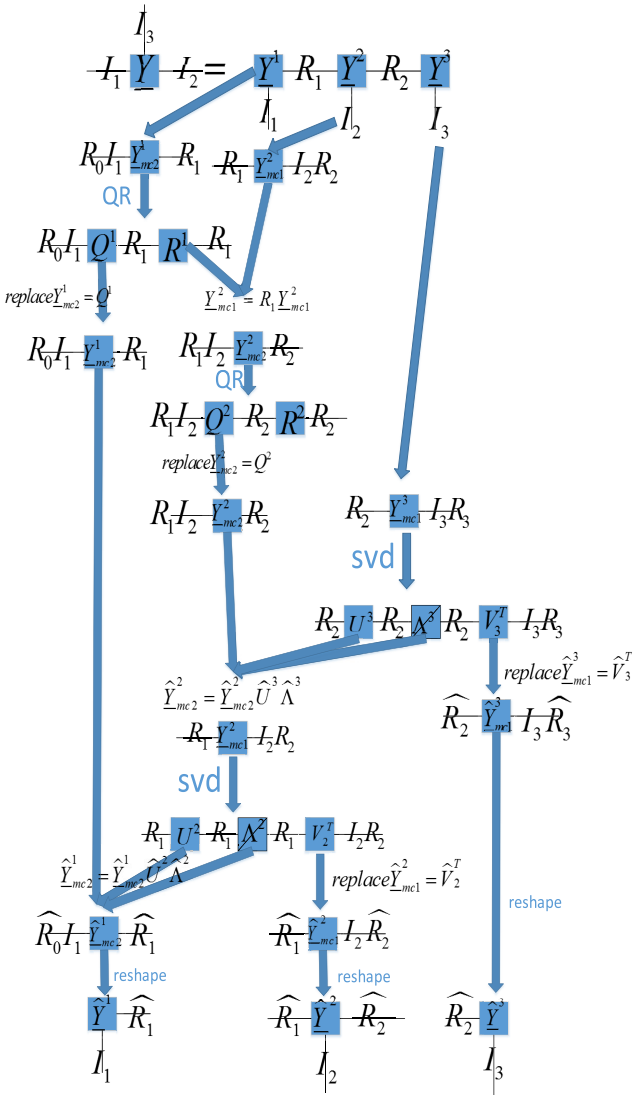

FIGURE 35. TT Truncation for a 3rd-order tensor. First perform TT decomposition on the original third-order tensor, (the rank of the TT decomposition at this time is relatively large) and then the TT approximate solution with the lower TT rank is found step by step according to the algorithm 9.

Algorithm 9 TT Truncation (I.V.Oseledets, 2011) [60]

Input:

The Nth-order tensor $\underline{Y} = \underline{Y}^1 \times_{3,1} \underline{Y}^2 \cdots \times_{3,1} \underline{Y}^N \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, the core tensor $\underline{Y}^n \in R^{R_{n-1} \times I_n \times R_n}$ with a large TT rank, $r_{TT}(\underline{Y}) = (R_1, R_2, \cdots, R_{N-1})$, $R_n = r(\underline{Y}_{mcn})$

Output:

Approximate tensor of TT decomposition $\widehat{\underline{Y}}$ with a small TT rank $r_{TT}(\widehat{\underline{Y}}) = \widehat{R}_1, \widehat{R}_2, \cdots, \widehat{R}_{N-1}, \widehat{R}_0 = \widehat{R}_N = 1, \widehat{R}_n = r(\widehat{\underline{Y}}_{mcn})$,such that $\|\underline{Y} - \widehat{\underline{Y}}\|_F \leq \varepsilon$

1: Initialize $\widehat{\underline{Y}} = \underline{Y}$, $a = \frac{\varepsilon}{\sqrt{N-1}}$;
2: **for** n=1 to N-1 **do**
3: $[Q^n, R^n] = QR - decomposition(\underline{Y}^n_{mc2})$,
4: Replace $\underline{Y}^n_{mc2} = Q^n$ and $\underline{Y}^{n+1}_{mc1} = R^n \underline{Y}^{n+1}_{mc1}$, $\underline{Y}^{n+1}_{mc1} \in R^{R_n \times I_{n+1} R_{n+1}}$
5: **end for**
6: **for** n=N to 2 **do**
7: $[U^n, \Lambda^n, V_n^T] = truncated - svd(\underline{Y}^n_{mc1}, a)$,
8: find the smallest rank $\widehat{R}_{n-1}$ such that $\sum_{i>\widehat{R}_{n-1}}^{R_{n-1}} \alpha_r^2 \leq a^2 \|\alpha\|_1 = a^2 (\sum_{i=1}^{R_{n-1}} |\alpha_i|)^2$;
9: Replace $\widehat{\underline{Y}}^{n-1}_{mc2} = \widehat{\underline{Y}}^{n-1}_{mc2} \widehat{U}^n \widehat{\Lambda}^n \in R^{\widehat{R}_{n-2} I_{n-1} \times \widehat{R}_{n-1}}$ and $\widehat{\underline{Y}}^n_{mc1} = \widehat{V}_n^T \in R^{\widehat{R}_{n-1} \times I_n \widehat{R}_n}$;
10: Reshape $\underline{Y}^n = \widehat{\underline{Y}}^n_{mc2}.reshape([\widehat{R}_{n-1}, I_n, \widehat{R}_n])$;
11: **end for**
12: **return** Approximate tensor $\widehat{\underline{Y}} = \widehat{\underline{Y}}^1 \times_{3,1} \widehat{\underline{Y}}^2 \cdots \times_{3,1} \widehat{\underline{Y}}^N \in R^{I_1 \times I_2 \times I_3 \times \cdots I_N}$, where $\widehat{\underline{Y}}^n \in R^{\widehat{R}_{n-1} \times I_n \times \widehat{R}_n}$, $\widehat{R}_0 = \widehat{R}_N = 1$

For HT decomposition, the utility is relatively poor due to the need to determine the binary tree, and most of us use TT decomposition. The biggest advantage of TT decomposition is that only the core tensor is used, and thus we just need to calculate between core tensors. However, as we mentioned earlier, one of the biggest drawbacks of TT decomposition is that if there is no lower TT rank (i.e., there is no low rank TT solution), the computational complexity will be high.

F. VARIOUS TYPES OF DECOMPOSITION APPLICATIONS

We can find that almost all tensor-based algorithms are inseparable from tensor decomposition because of huge amount of unknown parameters. Therefore, tensor decomposition becomes very important in high-dimensional problems. Next we will introduce some basic tensor decomposition applications.

As described in part two of this survey, we can find that rank-one decomposition can be applied in tensor regression to support the tensor and solve optimization problem with constraint terms. However, since not all tensors can be performed rank-one decomposition, its application has certain limitations. Some results can be seen in recent papers, such as Zhou *et al.*'s Tensor regression with applications in neuroimaging data analysis [55], Chen *et al.*'s A hierarchical support tensor machine structure for target detection on high-resolution remote sensing Images [45], Makantasis *et al.*'s Tensor-Based Classification Models for Hyperspectral Data Analysis [69], and Makantasis *et al.*'s Tensor-Based Nonlinear Classifier for High-Order Data Analysis [70].

For CP decomposition, the best approximate solution can usually be found even if there are no special constraints on the original tensor or factor (such as orthogonal, independent, sparse, etc.). Therefore, CP decomposition is applied in many tensor-based algorithms. Some results can be seen in recent papers, such as Tresp *et al.*'s Learning with memory embeddings [137], Biswas and Milanfar's Linear Support Tensor Machine With LSK Channels: Pedestrian Detection

in Thermal Infrared Images [121], Pham and Yan's Tensor Decomposition of Gait Dynamics in Parkinson's Disease [125], Xu *et al.*'s Application of support higher-order tensor machine in fault diagnosis of electric vehicle range-extender [147], Zdunek and Fonal Randomized Nonnegative Tensor Factorization for Feature Extraction from High-dimensional Signals [115], Kisil *et al.*'s Common and Individual Feature Extraction Using Tensor Decompositions: a Remedy for the Curse of Dimensionality? [57], and Kargas and Sidiropoulos's Completing a joint PMF from projections: A low-rank coupled tensor factorization approach [98].

In practice, we tend to impose constraints on the original input tensor or the resulting core tensor. So the application of Tucker decomposition is usually translated into the application of HOSVD decomposition. In fact, HOSVD decomposition is a multidimensional extension of PCA. Some results can be seen in recent papers, such as Hu *et al.*'s Attribute-Enhanced Face Recognition with Neural Tensor Fusion Networks [37], Fanaee-T and Gama's Tensor-based anomaly detection: An interdisciplinary survey [46], Chen *et al.*'s Robust supervised learning based on tensor network method [156], Sofuoglu and Aviyente's A Two-Stage Approach to Robust Tensor Decomposition [118], Imtia and Sarwate's Improved Algorithms for Differentially Private Orthogonal Tensor Decomposition [47], Kisil *et al.*'s Common and Individual Feature Extraction Using Tensor Decompositions: a Remedy for the Curse of Dimensionality? [57], and Kossaifi *et al.*'s Tensor Contraction Layers for Parsimonious Deep Nets [64].

Because of intuitive tree or chain representations, HT and TT decomposition are used in many places. However, because the tree structure is not necessarily unique, the HT decomposition always has a variety of tree structures. So researchers often extend HT decomposition to a fixed TT decomposition. For the traditional HT decomposition, please refer to Bachmayr *et al.*'s Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations [86], Zhang and Barzilay's Hierarchical low-rank tensors for multilingual transfer parsing [158], and Kountchev and Kountcheva's Truncated Hierarchical SVD for image sequences, represented as third order tensor [111]. In recent years, there have been many studies on TT decomposition, especially in terms of properties and algorithms. Here we give some references, such as Kressner and Uschmajew's On low-rank approximability of solutions to high-dimensional operator equations and eigenvalue problems [26], Steinlechner's Riemannian Optimization for Solving High-Dimensional Problems with Low-Rank Tensor Structure [92], Phan *et al.*'s Tensor networks for latent variable analysis. Part I: Algorithms for tensor train decomposition [10], Wu *et al.*. General tensor spectral co-clustering for higher-order data [130], Chen *et al.*'s Parallelized Tensor Train Learning of Polynomial Classifiers [161]. Wang *et al.*'s Support vector machine based on low-rank tensor train decomposition for big data applications [155], and Xu *et al.*'s Whole Brain fMRI Pattern Analysis Based on Tensor Neural Network [148].

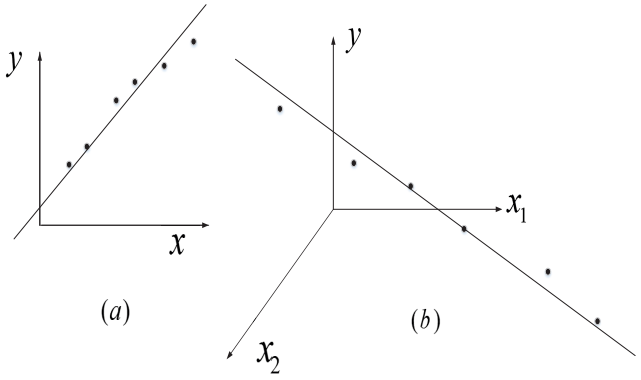

FIGURE 36. Traditional linear regression model. Where (a) is a linear regression of a two-dimensional plane, and (b) is a linear regression of three-dimensional space.

III. PART TWO: TENSOR APPLICATION IN MACHINE LEARNING AND DEEP LEARNING

The second part is based on the first part of tensor operation and tensor decomposition. This part mainly discusses the application of innovative algorithms for tensor in machine learning and deep learning. For example, converting the traditional input vector to a new tensor produces a new tensor-based algorithm, such as support tensor machine, tensor regression, and so on. These algorithms mainly achieve the goal of improving accuracy by finding structured information of the original data and performing subsequent data processing. Some algorithms tensorize weight matrix or vector, and use tensor decomposition on the resulting tensor. The number of elements can be reduced by tensor decomposition, which can effectively reduce the complexity and running time. Before the specific description, we give the outline of the algorithm (see table 1).

A. APPLICATION OF TENSOR IN REGRESSION

1) TENSOR REGRESSION

Consider a traditional linear regression model (see figure 36):

$$y = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b \tag{76}$$

where $\boldsymbol{x} \in R^N$ is sample feature vector, and $\boldsymbol{w} \in R^N$ is coefficient vector, b is bias. Regression models are often used to predict, such as stock market forecasts, weather forecasts, etc. When we expand the input $\boldsymbol{x}$ into a tensor, it becomes tensor regression. First let's consider a simple case where the input is a tensor $\underline{X} \in R^{I_1 \times I_2 \cdots I_N}$ and the predicted value y is a scalar. Usually tensor regression has the following expression:

$$y = \underline{W} \bullet \underline{X} + b \quad or \quad y = \underline{W} \bullet \underline{X} + b + \boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{c} \tag{77}$$

where $\underline{W} \in R^{I_1 \times I_2 \cdots I_N}$ is the coefficient vector, and b is the bias. Some researchers will sometimes add a vector-valued covariate $\boldsymbol{c}$. In general, the solution of tensor regression is to decompose the coefficient tensor and then solve factors by alternating least squares (ALS) method, such as rank-1 decomposition, CP decomposition, Tucker decomposition,

TABLE 1. Tensor based algorithm.

	Rank-one/CP	Tucker/HOSVD	HT/TT	others
Regression	Tensor regression (H. zhou, 2013) [55], Tensor regression(Hua zhou, 2013) [55],Tensor Learning for Regression(W. Guo, 2012) [141]	Tensor regression(R.Yu, 2016) [113], Generalized tensor regression (P.D.Hoff, 2015) [102]	Tensor representation of multivariate polynomial regression for scalar and vector variables (Z.Chen, 2018) [161]	Tensor variable Gaussian process regression (M. Hou, 2015) [90]
Supervised Classification	The Support Tensor Machine(D.Tao, 2005) [27], Tensor-Based Classification Models (K.Makantasis, 2018) [69], The Support Tensor Machine(I.Kotsia, 2012) [58]	Tensor-based graph embedding algorithm (Weiming Hu, 2017) [143], The Support Tensor Machine(I. Kotsia, 2011) [59], Tensor-based feature fusion for face recognition (Guosheng Hu, 2017) [37]	The Support Tensor Machine(Y. Wang, 2017) [155], Polynomial classifier algorithm based on tensor TT decomposition (Zhongming Chen, 2018) [161]	The Support Tensor Machine(Z.Hao, 2013) [164], Feature tensor generation (Tensorization) for image classification (Haoyu Yang, 2017) [51]
Data preprocessing	Tensor dictionary learning(Mohsen Ghassemi, 2017) [88], a tensor-based preprocessing of interictal epileptic EEG data (L. Albera, 2015) [76]	Feature Extraction for Incomplete Data Via Low-Rank Tensor Decomposition(Q. Shi, 2019) [107]	High-Order Tensor Completion for Data Recovery via Sparse Tensor-Train Optimization (L. Yuan, 2018) [83]	Exact Tensor Completion Using t-SVD(Z. Zhang, 2017) [165], a Preprocessing Method Based on Tensor Principal Component Analysis(Z. Chen, 2018) [160], Tensor Robust Principal Component Analysis(C. Lu, 2019) [20]
Unsupervised classification	High-order restricted Boltzmann machines(Nguyen et, 2015) [126]	An Intelligent Outlier Detection Method With One Class Support Tucker Machine(X. Deng, 2019) [145]	A Tensor-Train Deep Computation Model(Q. Zhang, 2018) [108]	latent conditional high-order Boltzmann machines(Yan Huang, 2017) [151]

TT decomposition, etc. For example, (Zhou *et al.*) [55] proposed the rank-1 and CP decomposition. Then the formula becomes:

$$y = \boldsymbol{w^1} \circ \boldsymbol{w^2} \circ \cdots \boldsymbol{w}^N \bullet \underline{X} + b + \boldsymbol{a^T c}$$
$$y = \underline{\Lambda} \times_{1m} W_1 \times_{2m} W_2 \cdots \times_{Nm} W_N \bullet \underline{X} + b + \boldsymbol{a^T c} \quad (78)$$

Tensor regression of the Tucker decomposition form is similar. For details, please refer to (Hoff *et al.* [102]; Yu *et al.* [113]). The general tensor regression is attributed to solving the following minimization problem:

$$L(a, b, \underline{W}) = \arg\min_{a,b,\underline{W}} \sum_{i=1}^{N} (\widehat{y}^i - y^i)^2, \quad i = 1, \cdots, N \quad (79)$$

where $\widehat{y}^i = \underline{W} \bullet \underline{X}^i + b + \boldsymbol{a^T c}$ represents the predicted value corresponding to the ith tensor sample, $\underline{X}^i$ represents the ith tensor sample, and y^i represents the true value of the ith tensor sample.

We give the following general algorithm for tensor regression (see algorithm 10).

2) TENSOR VARIABLE GAUSSIAN PROCESS REGRESSION

Tensor variable Gaussian process regression is similar to what we have introduced in the previous section. The same thing is that the input $\underline{X}^i \in R^{I_1 \times I_2 \cdots I_N}$ is still an Nth-order tensor and the output y^i is a scalar, and the difference is that the input here is subject to Gaussian distribution. (Hou *et al.*) [90] assumed that the output consists of a nonlinear function with respect to input X and Gaussian noise $\epsilon^i \sim N(0, \sigma^2)$, as follows:

$$y^i = f(\underline{X}^i) + \epsilon^i \quad i = 1, \cdots, N \quad (80)$$

The nonlinear function of the above formula can be modeled by a Gaussian process, as follows:

$$f(\underline{X}) \sim GP(m(\underline{X}), k(\underline{X}, \widetilde{\underline{X}})|\theta) \quad (81)$$

where $m(\underline{X})$ is the mean function, $k(\underline{X}, \widetilde{\underline{X}})$ is the kernel function and θ is the associated hyperparameter. For the sake of simplicity, we use the standard Gaussian process $m(\underline{X}) = 0$. For the kernel function, we use the product probability kernel:

$$k(\underline{X}, \widetilde{\underline{X}}) = \alpha^2 \prod_{n=1}^{N} exp(\frac{D[p(\boldsymbol{x}|\Omega_n^{\underline{X}})||q(\widetilde{\boldsymbol{x}}|\Omega_n^{\widetilde{\underline{X}}})]}{-2\beta_n^2}) \quad (82)$$

where α represents the amplitude parameter and β represents the scale parameter, $D(p||q) = \sum_{x=1} p(x) log \frac{px}{q(x)} = \int xp(x) log \frac{px}{q(x)} dx$ means the KL divergence, $p(\boldsymbol{x}|\Omega_n^{\underline{X}})$ means the Gaussian distribution of vector variable $\boldsymbol{x} = [x_1, \cdots, x_{I_d}]$, the mean vector and the covariance matrix are $\boldsymbol{\mu^n}$, $\sum^n$, respectively. Note that the mean vector and the covariance matrix are determined from the mode-n matricization $\underline{X}_{mn}$ of $\underline{X}$ by treating each $\underline{X}_{mn}$ as a probability model with I_n number of variables and $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \cdots \times I_N$ number of observations.

When we have determined the parameters from the training set, the purpose of the tensor Gaussian process regression is to infer the probability distribution of the output for the test point $\underline{X}_{test}$, i.e.:

$$p(y_{test}|\underline{X}_{test}, \underline{\boldsymbol{X}}, \boldsymbol{y}, \theta, \sigma^2) \quad (83)$$

where $\underline{\boldsymbol{X}} = [\underline{X}^1, \underline{X}^2, \cdots, \underline{X}^N]^T \in R^{N \times I_1 \times I_2 \cdots I_N}$ means combining all sample tensors, and $\boldsymbol{y} = [y^1, y^2, \cdots, y^N]^T \in R^N$. But actually we only need to know the distribution of $f(\underline{X}^{test})$

Algorithm 10 Tensor Regression Algorithm (Zhou *et al.*) [55]

Input:

N Nth-order sample data tensors $\underline{X}^i \in R^{I_1 \times I_2 \cdots I_N}$, $i = 1, \cdots, N$ and its true value y^i, a vector-valued covariate $\boldsymbol{c}$.;

Output:

$a, b, \underline{W}$;

1: Initialize $\underline{W} = 0$, solve $(a, b) = \min_{a,b,\underline{W}} \sum_{i=1}^{N} (\widehat{y}^i - y^i)^2$;
2: Initialize the factor matrices W_n for $n = 1, \cdots, N$ and core tensor $\underline{\Lambda}$ for CP decomposition or initialize the factor vectosr for rank-1 decomposition, other decomposition is similar;
3: **while** the number of iterations is not reached or there is no convergence **do**
4: **for** n=1 to N **do**
5: solve $W_n = \min_{W_n} L(a, b, \underline{\Lambda}, W_1, \cdots, W_{n-1}, W_{n+1}, \cdots, W_N)$;
6: **end for**
7: solve $\underline{\Lambda} = \min_{\underline{\Lambda}} L(a, b, \underline{\Lambda}, W_1, \cdots, W_N)$;
8: $(a, b) = \min_{a,b,\underline{W}} \sum_{i=1}^{N} (\widehat{y}^i - y^i)^2$;
9: **end while**

according to the expression. So it finally turns to solve the following expression:

$$p(f(\underline{X}^{test})|\underline{X}_{test}, \underline{\boldsymbol{X}}, \boldsymbol{y}, \theta, \sigma^2) \tag{84}$$

Here we omit the complicated calculations and give the results directly. It is noted that the test samples are also subject to the Gaussian distribution, and the probability properties of the distribution is accorded to Bayesian conditions. We get:

$$p(f(\underline{X}^{test})|\underline{X}_{test}, \underline{\boldsymbol{X}}, \boldsymbol{y}, \theta, \sigma^2) \sim N(\mu_{test}, \sigma^2_{test}) \tag{85}$$

where $\mu_{test} = \boldsymbol{k}(\underline{X}^{test}, \underline{\boldsymbol{X}})^T (K + \sigma^2 I)^{-1} \boldsymbol{y}$ and $\sigma^2_{test} = k(\underline{X}^{test}, \underline{X}^{test}) - \boldsymbol{k}(\underline{X}^{test}, \underline{\boldsymbol{X}})^T (K + \sigma^2 I)^{-1} \boldsymbol{k}(\underline{X}^{test}, \underline{\boldsymbol{X}})$.

Tensor variable Gaussian process regression is generally used to deal with noise-bearing and Gaussian-distributed data. It has certain limitations, and this method is computationally expensive. Without using tensor decomposition, the amount of parameter data is very large. Thus, the amount of calculation will also increase exponentially.

3) GENERALIZED TENSOR REGRESSION

Now we introduce a more general case where both input and output are tensors. We start with a simple second-order matrix. A second-order matrix regression is as follows:

$$Y^i = AX^iB^T + E \tag{86}$$

where $X^i \in R^{I_1 \times I_2}$, $Y^i \in R^{J_1 \times J_2}$, $i = 1, \cdots, N$ are N input sample matrices and corresponding output sample matrices. $A \in R^{J_1 \times I_1}$ and $B \in R^{J_2 \times I_2}$ are unknown coefficient matrices. $E \in R^{J_1 \times J_2}$ is a noise matrix with mean-zero.

(Hoff) [102] used the residual mean squared error to measure the error between the true value and the prediction value:

$$(A, B) = \arg\min_{A,B} \frac{\sum_{i=1}^{N} ||Y^i - AX^iB^T||_F^2}{n} \tag{87}$$

By deriving the above formula, we finally get:

$$\begin{aligned} A &= (\sum Y^iB(X^i)^T)(\sum X^iB^TB(X^i)^T)^{-1} \\ B &= (\sum (Y^i)^TAX^i)(\sum (X^i)^TA^TAX^i)^{-1} \end{aligned} \tag{88}$$

Similarly, we can get A and B respectively by alternating least squares.

We further extend to generalized tensor regression as follows:

$$\underline{Y}^i = \underline{X}^i \times_{1m} W_1 \times_{2m} W_2 \cdots \times_{nm} W_N + \underline{E} \tag{89}$$

where $W_n \in R^{J_n \times I_n}$ are coefficient matrices (factor matrices) and $\underline{X}^i \in R^{I_1 \times I_2 \cdots \times I_N}$, $\underline{Y}^i \in R^{J_1 \times J_2 \cdots \times J_N}$ are input and output tensors, respectively. $\underline{E} \in R^{J_1 \times J_2 \cdots \times J_N}$ is a Noise tensor.

Note that there is a property between the mode-n product and the Kronecker product, as follows:

$$\begin{aligned} \underline{Z} &= \underline{X} \times_{1m} W_1 \times_{2m} W_2 \cdots \times_{nm} W_N \\ \underline{Z}_{mn} &= W_n \underline{X}_{mn} (W_N \otimes_R \cdots \otimes_R W_{n+1} \\ &\quad \otimes_R W_{n-1} \cdots W_1)^T \\ \underline{Z}_{v1} &= (W_N \otimes_R \cdots \otimes_R W_1) \underline{X}_{v1} \end{aligned} \tag{90}$$

Therefore, we only need to adopt the mode-n matricization on both sides of the formula 89 to get the solution:

$$\underline{Y}_{mn} = W_n \widetilde{\underline{X}}_{mn} + \underline{E}_{mn} \tag{91}$$

where $mat(\widetilde{\underline{X}})_n = mat(\underline{X})_n (W_N \otimes_R \cdots \otimes W_{n+1} \otimes_R W_{n-1} \cdots W_1)^T$. Then through formula 88 we finally get:

$$W_n = (\sum \underline{Y}^i_{mn} (\widetilde{\underline{X}}^i_{mn})^T) \quad (\sum \widetilde{\underline{X}}^i_{mn} (\widetilde{\underline{X}}^i_{mn})^T)^{-1} \tag{92}$$

Finally, we give the specific algorithm of the whole generalized tensor regression (see algorithm 11).

Algorithm 11 Generalized Tensor Regression (Hoff) [102]

Input:

N Nth-order sample data tensors $\underline{X}^i \in R^{I_1 \times I_2 \cdots I_N}$, $i = 1, \cdots, N$ and output tensor $Y^i \in R^{J_1 \times J_2}$,;

Output:

$W_n, n = 1, \cdots, N$;

1: Initialize W^n as random matrices;
2: **while** the number of iterations is not reached or there is no convergence **do**
3: **for** n=1 to N **do**
4: Calculate W_n by formula 92;
5: **end for**
6: **return** W_n;
7: **end while**

4) TENSOR REPRESENTATION OF MULTIVARIATE POLYNOMIAL REGRESSION FOR SCALAR VARIABLES

Multivariate polynomial regression is a generalization of linear regression and multiple regression, which predicts the next moment or possible future output by processing the interaction between multiple variables. We first consider a simple binary quadratic regression:

$$y = a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2 + a_{11}x_1^2 + a_{22}x_2^2 \tag{93}$$

One of the coefficients a_{12} represents the relationship between two variables. We can use the multiplication of matrix and vector to represent the above formula:

$$y = \begin{bmatrix}1 & x_1 & x_1^2\end{bmatrix}\begin{bmatrix}a_0 & a_2 & a_{22}\\ a_1 & a_{12} & 0\\ a_{11} & 0 & 0\end{bmatrix}\begin{bmatrix}1\\ x_2\\ x_2^2\end{bmatrix} \tag{94}$$

Consider a more complex complete binary quadratic polynomial regression (or more general binary quadratic polynomial regression):

$$\begin{aligned}y = {} & a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2 + a_{11}x_1^2 + a_{22}x_2^2\\ & + a_{112}x_1^2x_2 + a_{122}x_1x_2^2 + a_{1122}x_1^2x_2^2\end{aligned} \tag{95}$$

We can also use a similar product form of matrix and vector, as follows:

$$y = \begin{bmatrix}1 & x_1 & x_1^2\end{bmatrix}\begin{bmatrix}a_0 & a_2 & a_{22}\\ a_1 & a_{12} & a_{122}\\ a_{11} & a_{112} & a_{1122}\end{bmatrix}\begin{bmatrix}1\\ x_2\\ x_2^2\end{bmatrix} \tag{96}$$

Using the mode-n product property of tensor and vector, the above equation can be transformed into:

$$y = \underline{A} \times_{1v} \begin{bmatrix}1\\ x_1\end{bmatrix} \times_{2v} \begin{bmatrix}1\\ x_1\end{bmatrix} \times_{3v} \begin{bmatrix}1\\ x_2\end{bmatrix} \times_{4v} \begin{bmatrix}1\\ x_2\end{bmatrix} \tag{97}$$

where the 4th-order tensor $\underline{A} \in R^{2\times2\times2\times2}$ is the coefficient tensor and :

$$\underline{A} = \left[\begin{bmatrix}\begin{bmatrix}a_0 & \frac{1}{2}a_2\\ \frac{1}{2}a_2 & a_{22}\end{bmatrix}, \begin{bmatrix}\frac{1}{2}a_1 & \frac{1}{4}a_{12}\\ \frac{1}{4}a_{12} & \frac{1}{2}a_{122}\end{bmatrix}\end{bmatrix}, \begin{bmatrix}\begin{bmatrix}\frac{1}{2}a_1 & \frac{1}{4}a_{12}\\ \frac{1}{4}a_{12} & \frac{1}{2}a_{122}\end{bmatrix}, \begin{bmatrix}a_{11} & \frac{1}{2}a_{112}\\ \frac{1}{2}a_{112} & a_{1122}\end{bmatrix}\end{bmatrix}\right] \tag{98}$$

For a general case of N variables, the complete multivariate polynomial regression can be expressed as follows (Chen and Billings) [117]:

$$\begin{aligned}y &= \sum_{i_1=0}^{N}\cdots\sum_{i_N=0}^{N} a_{i_1i_2\cdots i_N}x_1^{i_1}x_2^{i_2}\cdots x_N^{i_N}\\ &= \underline{A} \times_{1v} \begin{bmatrix}1\\ x_1\end{bmatrix}\cdots\times_{Nv}\begin{bmatrix}1\\ x_1\end{bmatrix} \times_{(N+1)v}\begin{bmatrix}1\\ x_2\end{bmatrix}\cdots\times_{2Nv}\\ &\begin{bmatrix}1\\ x_2\end{bmatrix}\cdots\times_{(N^2-N+1)v}\begin{bmatrix}1\\ x_N\end{bmatrix}\cdots\times_{(N^2)v}\begin{bmatrix}1\\ x_N\end{bmatrix}\\ &= \underline{B} \times_{1v} V(x_1) \times_{2v} V(x_2)\cdots\times_{Nv} V(x_N)\end{aligned} \tag{99}$$

where $\underline{A}$ is an N^2th-order tensor with size $2\times2\times\cdots\times2$, $\underline{B}$ is an Nth-order tensor with size $(N+1)\times(N+1)\times\cdots\times(N+1)$, and $V(x_n)$ is the Vandermonde vector of x_n:

$$V(x_n) = \begin{bmatrix}1 & x_n & x_n^2\cdots & x_n^N\end{bmatrix}^T \tag{100}$$

Because of its breadth and generality, this model is used in many fields, such as weather prediction, stock forecasting and other regression models. But we can clearly see that as the variables increase, the unknown coefficients will rise exponentially, which will greatly increase the complexity. So we need to reduce unknown parameters by low-rank tensor decomposition network. (Chen *et al.*) [159] decomposed the coefficient tensor with low rank TT decomposition. Another way is to use a truncation model, which is similar to the coefficient tensor of the binary case just mentioned. It only takes two elements for each dimension of $\underline{B}$, so the truncated expression is as follows:

$$y = \underline{B}_t \times_{1v} \begin{bmatrix}1\\ x_1\end{bmatrix} \times_{2v} \begin{bmatrix}1\\ x_2\end{bmatrix}\cdots\times_{Nv}\begin{bmatrix}1\\ x_N\end{bmatrix} \tag{101}$$

where $\underline{B}_t$ is an Nth-order truncated tensor of size $2\times\cdots\times2$. However, the coefficient tensor $\underline{B}$ in the second term of formula 108 is the N^2th-order tensor. This is the Nth-order tensor. The calculation can be simplified greatly, and the subsequent duplicates are also reduced by using a new truncated tensor.

5) TENSOR REPRESENTATION OF MULTIVARIATE POLYNOMIAL REGRESSION FOR VECTOR VARIABLES

In the previous section, we introduced the multivariate polynomial regression of traditional scalar variables. Then we extend to the vector form. Here we directly give the general vector form of the generalized (complete) binary quadratic regression, which is similar to formula 95, as follows:

$$\begin{aligned}y = {} & a_0 + \boldsymbol{a}_1^T\boldsymbol{x}_1 + \boldsymbol{a}_2^T\boldsymbol{x}_2 \mathcal{C} \boldsymbol{x}_1^T\boldsymbol{A}_{12}\boldsymbol{x}_2 \mathcal{C} \boldsymbol{x}_1^T\boldsymbol{A}_{11}\boldsymbol{x}_1\\ & + \boldsymbol{x}_2^T\boldsymbol{A}_{22}\boldsymbol{x}_2 + \underline{\boldsymbol{A}}_{\boldsymbol{112}} \times_{1v} \boldsymbol{x}_1 \times_{2v} \boldsymbol{x}_1 \times_{3v} \boldsymbol{x}_2\\ & + \underline{\boldsymbol{A}}_{\boldsymbol{122}} \times_{1v} \boldsymbol{x}_1 \times_{2v} \boldsymbol{x}_2 \times_{3v} \boldsymbol{x}_2\\ & + \underline{\boldsymbol{A}}_{\boldsymbol{1122}} \times_{1v} \boldsymbol{x}_1 \times_{2v} \boldsymbol{x}_2 \times_{3v} \boldsymbol{x}_2 \times_{4v} \boldsymbol{x}_2\end{aligned} \tag{102}$$

where $\boldsymbol{x_1}, \boldsymbol{x_2}$ represent vectors, $\boldsymbol{A_{11}}, \boldsymbol{A_{12}}, \boldsymbol{A_{22}}$ represent matrices. $\underline{\boldsymbol{A}}_{\boldsymbol{112}}, \underline{\boldsymbol{A}}_{\boldsymbol{122}}$ are 3rd-order tensors and $\underline{\boldsymbol{A}}_{\boldsymbol{1122}}$ is a 4th-order tensor.

We directly give the equivalent tensor product form of the above formula, as follows:

$$\begin{aligned}y &= w_0 + \sum_{n=1}^{N^2}\sum_{i_1,i_2,\cdots,i_n=1}^{N} \underline{\boldsymbol{A}}_{i_1,i_2,\cdots,i_n} \times_{1v} \boldsymbol{x_{i_1}} \cdots \times_{nv} \boldsymbol{x_{i_n}}\\ &= \underline{\boldsymbol{B}} \times_{1v} \begin{bmatrix}1\\ \boldsymbol{x_1}\end{bmatrix}\cdots\times_{Nv}\begin{bmatrix}1\\ \boldsymbol{x_1}\end{bmatrix} \times_{(N+1)v}\begin{bmatrix}1\\ \boldsymbol{x_2}\end{bmatrix}\\ &\quad \cdots\times_{2Nv}\begin{bmatrix}1\\ \boldsymbol{x_2}\end{bmatrix}\cdots\times_{(N^2-N+1)v}\begin{bmatrix}1\\ \boldsymbol{x_N}\end{bmatrix}\cdots\times_{(N^2)v}\begin{bmatrix}1\\ \boldsymbol{x_N}\end{bmatrix}\\ &= \underline{\boldsymbol{C}} \times_{1v} \boldsymbol{V(x_1)} \times_{2v} \boldsymbol{V(x_2)}\cdots\times_{Nv} \boldsymbol{V(x_N)}\end{aligned} \tag{103}$$

where $\boldsymbol{x_n} \in I_n$, $\underline{\boldsymbol{A}}_{i_1,i_2,\cdots,i_n}$ are Nth-order tensors($n \in [1, N^2]$) of size $I_{i_1} \times I_{i_2}\cdots\times I_{i_n}$, $\underline{\boldsymbol{B}}$ is an Nth-order tensor with

size $B_1 \times B_2 \times \cdots \times B_N$, where $B_n = \frac{(I_n)^{N+1}-1}{I_n-1}$, $\underline{\boldsymbol{C}}$ is an N^2th-order tensor with size $(I_1+1)\times\cdots\times(I_1+1)\times(I_2+1)\times\cdots\times(I_2+1)\cdots\times(I_N+1)\cdots\times(I_N+1)$, and $\boldsymbol{V}(\boldsymbol{x_n})$ is the Vandermonde vector of x_n:

$$\boldsymbol{V}(\boldsymbol{x_n}) = \begin{bmatrix} 1 & \boldsymbol{x_n}^T & (\boldsymbol{x_n}\otimes\boldsymbol{x_n})^T\cdots & (\boldsymbol{x_n}\otimes\cdots\otimes\boldsymbol{x_n})^T \end{bmatrix}^T \tag{104}$$

This model can be generalized to multidimensional tensors. Similar to the scalar form of multivariate polynomial regression, the multivariate polynomial regression in the form of vector (tensor) also has an exponential rise in complexity as the variable n increases. Similarly, we can reduce the coefficient tensor from N^2th-order to Nth-order. We can also use CP decomposition, Tucker decomposition, or TT decomposition to get a truncated model. Please refer to (Stoudenmire andSchwab,2016 [30]; Cohen andShashua, 2016 [94]) for details.

6) DISCUSSION AND COMPARISON

This section mainly introduces five tensor regressions. We extend from the simplest model to the multivariate regression of the most complex vector variable values. Usually the first and third tensor regressions are more common. For the first tensor regression, since the factors obtained by the rank 1 decomposition are all vectors, the calculation is relatively simple and the complexity is low. The tensor regression based on CP decomposition is slightly different from the tensor regression based on Tucker decomposition. When given tensor rank, CP decomposition is unique, and Tucker decomposition is usually not unique. Since the factors after CP decomposition are all matrices, in general, the factor after Tucker decomposition still has core tensor, so CP decomposition is still much simpler in performing operations. Since the factors obtained by CP decomposition is usually unique, and the factors of Tucker decomposition is usually not unique. For Tucker regression, we can choose the most accurate ones from many weighted tensors. Therefore, the Tucker regression is better than CP regression in terms of the accuracy. For multivariate generalized regression scenarios, tensors are used instead of complex coefficients. The coefficient tensor decomposition not only reduces the complexity, but also clearly expresses the structural relationship between the data.

B. APPLICATION OF TENSOR IN CLASSIFICATION

1) SUPPORT TENSOR MACHINE(STM) APPLICATION IN IMAGES CLASSFICATION

a: THE SUPPORT VECTOR MACHINE(SVM)

First, we briefly review the concept of support vector machine(SVM). The SVM is first proposed by (Corts and Vapnik) [18] to find a hyperplane to distinguish between the two different categories, what we usually call the binary classifier (see figure 37).

As can be seen from figure 37, the purpose of the SVM is to find a hyperplane $\boldsymbol{w^T x} + b = 0, \boldsymbol{x} = [x_1, x_2, \cdots, x_m]$ to distinguish between the two classes. We give the two types of labels $+1$ and -1 respectively. Where the distance from a point x to the hyperplane in the sample space is:

$$d = \frac{|\boldsymbol{w^T x} + b|}{\|\boldsymbol{w}\|} \tag{105}$$

As shown in figure 37, the point closest to the hyperplane is called the support vector, and the sum of the distances of the two heterogeneous support vectors to the hyperplane is:

$$\gamma = \frac{2}{\|\boldsymbol{w}\|} \tag{106}$$

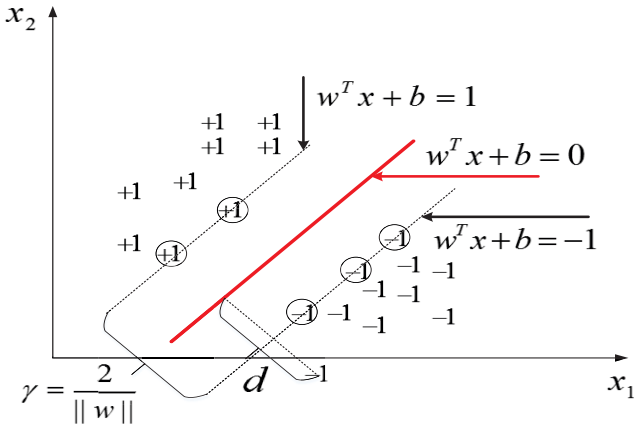

FIGURE 37. A simple schematic of a linear SVM. As shown, the input is a first-order tensor (vector) and the size is 2.

It is also called the margin. In order to find the hyperplane with the largest interval, it is converted to solve the following optimization problem:

$$\begin{aligned} &\max_{w,b} \ \frac{2}{\|\boldsymbol{w}\|} \\ &s.t. \quad y_i(\boldsymbol{w^T x_i}\ \boldsymbol{\mathcal{C}}\ \boldsymbol{b}) \geq 1, \quad i = 1\cdots, m \end{aligned} \tag{107}$$

where $\|\boldsymbol{w}\|$ is the two norm of the vector $\boldsymbol{w}$.

In fact, the training samples are linearly inseparable in many cases, which is called the soft interval. The general constraint formula for the SVM is shown as follows:

$$\begin{aligned} &\max_{w,b,\xi_j} \ \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{j=1}^{M}\xi_j \\ &s.t. \quad y_j(\boldsymbol{w^T x_j}\ \boldsymbol{\mathcal{C}}\ \boldsymbol{b}) \geq 1-\xi_j, \xi_j \geq 0, \quad j = 1, 2\cdots, M. \end{aligned} \tag{108}$$

where $\xi_j = l(y_i(\boldsymbol{w^T x_i} + b) - 1)$ is called slack variables. l is a loss function. There are three commonly used loss functions

$$\begin{aligned} hinge\ loss: \ l_{hinge}(x) &= max(0, 1-x); \\ exponential\ loss: \ l_{exp}(x) &= exp(-x); \\ logistic\ loss: \ l_{log}(x) &= log(1+exp(-x)); \end{aligned} \tag{109}$$

Later researchers (Zhao *et al.*) [144] converted the above constraints into the following formula:

$$\min_{w,b,\xi_j} \frac{\|\boldsymbol{w}\|^2}{2} + \frac{\gamma}{2}\boldsymbol{\xi}^{\boldsymbol{T}}\boldsymbol{\xi}$$
$$s.t. \quad y_j(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x_j}\ \boldsymbol{\mathcal{C}}\ \boldsymbol{b}) = 1 - \xi_j, \xi_j \geq 0, \quad j = 1, 2 \cdots, M. \tag{110}$$

where $\boldsymbol{\xi} = [\xi_1, \xi_2, \cdots, \xi_M] \in R^M$. Note that formula 110 has two major differences compared to formula 108. 1: in order to facilitate the calculation, the above constraint is changed from inequality to equality. 2: the loss function in formula 110 is the mean square loss. The benefit of this modification is that the solution will be easier. Generally, the solution is developed by Lagrangian multiplier method. We do not repeated derivation here. For details, please refer to (Corts and Vapnik) [18].

b: THE SUPPORT MATRIX MACHINE(SMM)

If we extend the input sample from vector to second-order tensor (matrix), we will get the Support Matrix Machine(SMM). (Luo) [80] proposed the concept of the Support Matrix Machine. We consider a matrix sample $X_a \in R^{I\times J}, a = 1, 2, \cdots, m$. The hinge loss function are replaced in SMM. The following constraint formula is obtained:

$$\min_{W,b,\xi_j} \frac{1}{2}tr(W^TW) + C\sum_{j=1}^{M} max(0, 1 - y_j[tr(W^TX_j) + b]) + \lambda\|W\|_*$$
$$s.t.\ y_j[tr(W^TX_j) + b] \geq 1 - max(0, 1 - y_j[tr(W^TX_j) + b]). \tag{111}$$

where $\|W\|_*$(we usually call it the nuclear norm) represents the sum of all singular values of the matrix W, C and λ are coefficient. In fact, we get the following properties after performing the mode-1 vectorization of the matrix $\boldsymbol{w} = vec(W^T)_1$.

$$tr(W^TX_j) = vec(W^T)_1^T vec(X_j^T)_1 = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x_i}$$
$$tr(W^TW) = vec(W^T)_1^T vec(W^T)_1 = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{w}. \tag{112}$$

Substituting the formula 136 into the formula 135 returns the constraint expression of the original SVM. Note that in order to protect the data structure from being destroyed, we generally do not perform the mode-n vectorization of the matrix and convert it into a traditional SVM. So we give the optimization problem directly in the form of a matrix. According to (Goldstein *et al.*) [39], they further converted the above constraints into the following augmented Lagrangian function form:

$$L(W, b, S, \lambda) = F(W, b) + G(S) + tr[\Lambda^T(S - W)] + \frac{a}{2}\|S - W\|_F^2,\ a\ is\ hyperparameter \tag{113}$$

where $F(W, b) = \frac{1}{2}tr(W^TW) + C\sum_{j=1}^{m} max(0, 1 - y_j[tr(W^TX_j) + b])$, $G(S) = \lambda\|W\|_*$. Due to the complexity of the SMM solution, please refer to (Luo *et al.*) [80] for details.

c: THE SUPPORT TENSOR MACHINE(STM)

If we further extend the matrix to tensor, we will get the Support Tensor Machine(STM). In general, STM currently have five constraint expressions, we first give the original constraint expression:

$$\max_{w,b,\xi_j} \frac{\|\underline{W}\|^2}{2} + C\sum_{j=1}^{M}\xi_j$$
$$s.t. \quad y_j(\underline{W} \bullet \underline{X}_j + b) \geq 1 - \xi_j\xi_j \geq 0, \quad j = 1, 2 \cdots, M. \tag{114}$$

Here we usually choose to decompose the coefficient tensor $\underline{W}$, and the researchers give four solutions in total. (Tao *et al.*) [27] proposed to decompose the coefficient tensor into the form of the rank-one vector outer product, i.e., $\underline{W} = w^1 \circ w^2 \circ \cdots w^N$ (see formula 28). (Kotsia *et al.*) [58] performed CP decomposition on the coefficient tensor, i.e., $\underline{W} = \sum_{r=1}^{R}\lambda_r w^1 \circ w^2 \circ \cdots w^N$ (see formula 29). (Kotsia and Patras) [59] performed Tucker decomposition on the coefficient tensor, i.e., $\underline{W} = \underline{A} \times_{1m} W_1 \times_{2m} W_2 \cdots \times_{Nm} W_N$ (see formula 108). (Wang *et al.*) [155] performed TT decomposition on the coefficient tensor, i.e., $\underline{W} = \underline{W}_1 \times_{3,1} \underline{W}_2 \cdots \times_{3,1} \underline{W}_N$ (see formula 55). Substituting these three decompositions will result in three forms of STM.

In general, the solution of STM is similar to the solution of CP decomposition. The central idea is based on the alternating least squares method, that is, N-1 other optimization items are fixed first, and only one item is updated at a time. For example, if we use the form of the rank-one decomposition for coefficient tensor, then the constraint expression becomes as follows (see algorithm 12):

$$\max_{w_m,b,\xi_j} \frac{1}{2}\|w_m\|^2\alpha + C\sum_{j=1}^{M}\xi_j$$
$$s.t. \quad y_j(w_m^T(\underline{X}_j \times_{(i\neq m)v} w_i) + b) \geq 1 - \xi_j,$$
$$i = 1, 2 \cdots, n-1, n+1, \cdots, N.j = 1, 2 \cdots, M. \tag{115}$$

where $\alpha = \|\circ_{i=1,i\neq m}^{N} w_i\|^2$.

Then the label of a test sample, $\underline{X}_{test}$, can be predicted as follows:

$$y = sign(\underline{X}_{test} \times_{1v} w_1 \cdots \times_{Nv} w_N + b) \tag{116}$$

However, the above-mentioned alternating least squares iteration method usually needs a lot of time and computational memory, and only obtian a local optimal solution. So many researchers proposed other algorithms.

Algorithm 12 Support Tensor Machine (Hao *et al.*) [164]

Input:

Input tensor sample sets $\underline{X}_j \in R^{I_1 \times I_2 \cdots I_N}, j = 1, 2 \cdots, M$ and label $y_j \in \{+1, -1\}$;

Output:

$w_i, i = 1, \cdots, N$ and b;

1: **if** the required number of iterations is not reached **then**
2: **for** m=1 to N **do**
3: Initialize $w_1, w_2, \cdots, w_{m-1}, w_{m+1}, \cdots, w_N$
4: $\alpha = \| \circ_{i=1, i \neq m}^{N} w_i \|^2$
5: Calculate w_m by solving the binary optimization problem of the formula 139.
6: **end for**
7: **end if**

(Z.Hao, 2013) [164] proposed to transform the formula 114 into the following constraint expression:

$$\min_{\alpha} \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_i y_i y_j (\underline{X}_i \bullet \underline{X}_j) - \sum_{j=1}^{M} \alpha_j$$
$$s.t. \quad \sum_{j=1}^{M} \alpha_j y_j = 0, \ 0 \leq \alpha_j \leq C \quad j = 1, 2 \cdots, M. \quad (117)$$

where α_j are the Lagrange multipliers. Note that if the input tensor becomes a vector, formula 117 will become the dual problem of the standard SVM.

STM has gradually entered the field of machine learning due to its ability to preserve data structures and improve performance. STM with different constraints have separate application scenarios. For example, STM based on CP decomposition is applied to pedestrian detection of thermal infrared rays in order to find pedestrians in a group of images for precise positioning (Biswas and Milanfar) [121]. STM based on rank-one decomposition is applied to high-resolution remote sensing image target detection (Chen *et al.*) [45]. STM based on the original dual problem solving algorithm is applied to the fault diagnosis of electric vehicle range finder (Xu *et al.*) [147].

2) HIGH-ORDER RESTRICTED BOLTZMANN MACHINES (HORBM) FOR CLASSIFICATION

We first review the concept of Restricted Boltzmann Machines. RBM is a random neural network, which can be used for algorithm modeling of dimensionality reduction, classification, collaborative filtering, etc. In RBM, it contains two layers, including visible layer and hidden layer (see figure 38). Where the visible layer is, $\boldsymbol{x} = [x_1, x_2, \cdots, x_M]^T \in R^M$, hidden layer is, $\boldsymbol{y} = [y_1, y_2, \cdots, y_N]^T \in R^N$, $x_m = \{0, 1\}$, $m \in [1, M]$ and $y_n = \{0, 1\}$, $m \in [1, N]$. The weight of the interconnection is $W \in R^{M \times N}$. The visible layer has a bias of $\boldsymbol{a} \in R^M$ and the hidden layer has a bias of $\boldsymbol{b} \in R^N$. When the input value of the visible layer is given, the value of the hidden layer is

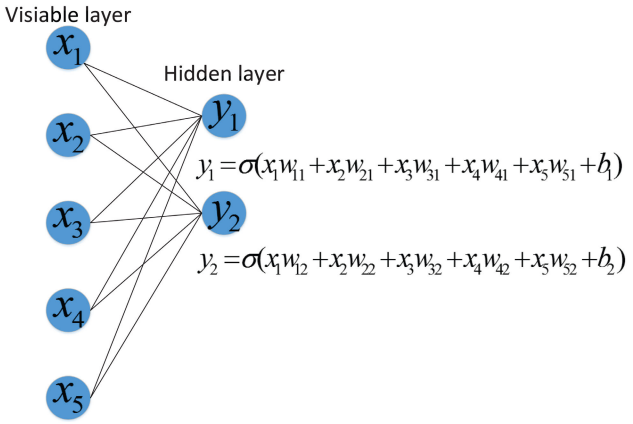

FIGURE 38. A simple schematic of the RBM, with the visible layer variable $x = [x_1, \cdots, x_5]$ on the left and the hidden layer variable $y = [y_1, y_2]$ on the right.

derived from the following formula:

$$\boldsymbol{y} = \sigma(W^T \boldsymbol{x} + \boldsymbol{b}) \quad (118)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the activation function.

Then we carry out the back propagation algorithm, which recalculates the value of the visible layer as the input of the hidden layer's value:

$$\boldsymbol{x} = \sigma(W\boldsymbol{y} + \boldsymbol{a}) \quad (119)$$

When the back-propagation recalculated visible layer value is not equal to the original visible layer value, the operation is repeated, which is the training process of the restricted Boltzmann machine. KL divergence is usually used in a Restricted Boltzmann Machine to measure the distance between the distributions of these two variables. RBM is a probability distribution model based on energy, as follows:

$$E(\boldsymbol{x}, \boldsymbol{y}) = -\boldsymbol{a}^T \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{y} - \boldsymbol{y}^T W \boldsymbol{x} \quad (120)$$

Then we derive the joint probability distribution of the hidden layer variable y and the visible layer variable x:

$$P(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{Z} e^{-E(\boldsymbol{x}, \boldsymbol{y})} \quad (121)$$

where $Z = \sum_{\boldsymbol{x}, \boldsymbol{y}}$. In order to make the distribution of these two values as close as possible to maximize the likelihood function of the input samples:

$$L(W, \boldsymbol{a}, \boldsymbol{b}) = argmax \ E[\sum_{t=1}^{N} log \ P(\boldsymbol{x}^t)]$$
$$P(\boldsymbol{x}) = \sum_{\boldsymbol{y}} P(\boldsymbol{x}, \boldsymbol{y}) \quad (122)$$

We assume that there are N input samples, $x^t \in R^M, t \in [1, N]$ in visible layer. Since the derivative form of the above formula cannot be solved generally, the deep learning pioneer Hinton proposed the CD algorithm (i.e.,k times Gibbs sampling) to obtain an approximate solution. Here we give a

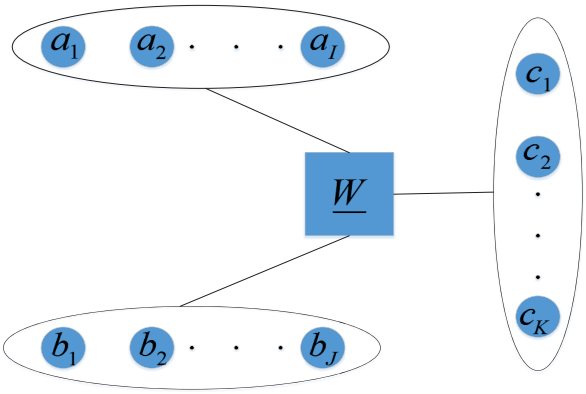

FIGURE 39. Schematic diagram of the energy function of the three sets of variables. The middle is the weight tensor, the above is the variable a, the lower is the variable b, and the right is the variable c.

very simple update formula based on the actual application. In practice, it usually takes only one sample to achieve very accurate results, so the updated formula is as follows:

$$\begin{aligned} W &= W + \alpha(\boldsymbol{x}\boldsymbol{y} - \boldsymbol{x}^{\boldsymbol{1}}\boldsymbol{y}^{\boldsymbol{1}}) \\ \boldsymbol{a} &= \boldsymbol{a} + \alpha(\boldsymbol{x} - \boldsymbol{x}^{\boldsymbol{1}}) \\ \boldsymbol{b} &= \boldsymbol{b} + \alpha(\boldsymbol{y} - \boldsymbol{y}^{\boldsymbol{1}}) \end{aligned} \tag{123}$$

where $\alpha \in [0, 1]$ is the learning rate, $\boldsymbol{x}^{\boldsymbol{1}}$ is the updated value of the visible layer variable $\boldsymbol{x}$ obtained by the first back-propagation of the hidden layer $\boldsymbol{y}$, and y^1 is the first update value of the hidden layer obtained by $\boldsymbol{x}^{\boldsymbol{1}}$ forward propagation again. If it is $k(k > 1)$ times, we only need to change the $\boldsymbol{x}^{\boldsymbol{1}}$ of the above formula to $\boldsymbol{x}^{\boldsymbol{k}}$ (the value of the visible layer variable obtained by the kth back-propagation). For details, please refer to (Hinton) [35].

If we increase the number of layers, the traditional RBM will become a higher dimension, which we call **High-order restricted Boltzmann machines (HORBM)**. For example, for three sets of variables, $\boldsymbol{a} \in R^I, \boldsymbol{b} \in R^J, \boldsymbol{c} \in R^K$, the energy function can be represented (see figure 39):

$$\begin{aligned} E(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) &= -\sum_{i,j,k=1}^{I,J,K} w_{i,j,k} a_i b_j c_k - \boldsymbol{d}^{\boldsymbol{T}}\boldsymbol{a} - \boldsymbol{e}^{\boldsymbol{T}}\boldsymbol{b} - \boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{c} \\ &= \underline{W} \times_{1v} \boldsymbol{a} \times_{2v} \boldsymbol{b} \times_{3v} \boldsymbol{c} - \boldsymbol{d}^{\boldsymbol{T}}\boldsymbol{a} - \boldsymbol{e}^{\boldsymbol{T}}\boldsymbol{b} - \boldsymbol{f}^{\boldsymbol{T}}\boldsymbol{c} \end{aligned} \tag{124}$$

where $\boldsymbol{a} \in R^I$ and $\boldsymbol{b} \in R^J$ are two input variables, which can be understood as two visible layers, $\boldsymbol{c} \in R^K$ is a hidden layer variable, and $\boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}$ correspond to the biases of three variables.

Note that the input of the visible, hidden layer or the additional layer of the RBM is a vector. If the input becomes a tensor, we call it **Tensor-variate Restricted Boltzmann machines (TvRBMs)** (Nguyen *et al.*) [126]. We assume that the visible layer variable is, $\underline{X} \in R^{I_1 \times I_2 \cdots \times I_N}$, and the hidden layer variable is, $\boldsymbol{y} \in R^J$, so the weight tensor is $\underline{W} \in R^{I_1 \times I_2 \cdots \times I_N \times J}$. Then the energy function can be similarly expressed as follows:

$$E(\underline{X}, \boldsymbol{y}) = \underline{A} \bullet \underline{X} - \boldsymbol{b}^{\boldsymbol{T}}\boldsymbol{y} - \underline{W} \bullet (\underline{X} \circ \boldsymbol{y}) \tag{125}$$

where $\underline{A} \in R^{I_1 \times I_2 \cdots \times I_N}, \boldsymbol{b} \in R^J$ are the biases of the visible and hidden layers, respectively. And similarly, the hidden layer variable $\boldsymbol{y} = [y_1, \cdots, y_J]^T$ can be expressed as:

$$y_j = \sigma(\underline{X} \bullet \underline{W}(:, \cdots, :, j) + b_j), \quad j = 1, 2 \cdots, J \tag{126}$$

A major problem is that as the input tensor dimension increases, the weight tensor elements will multiply. We usually use low rank tensor decomposition to solve the problem. For example, if we perform CP decomposition on weight tensors:

$$\underline{W} \approx \underline{\Lambda} \times_{1m} W_1 \times_{2m} W_2 \cdots \times_{Nm} W_N \times_{(N+1)m} W_{N+1} \tag{127}$$

where $W_n \in R^{I_n \times R}, n = 1, \cdots, N, W_{N+1} \in R^{J \times R}$ are factor matrices, and $\underline{\Lambda}$ is the diagonal tensor. Then the number of elements is reduced from the original $J \prod_{n=1}^{N} I_n$ to $R(J + \sum_{n=1}^{N} I_n + 1)$.

More simply, if the weight tensor can be expressed in the form of a rank-one vector outer product:

$$\underline{W} = \boldsymbol{w}^1 \circ \boldsymbol{w}^2 \circ \cdots \boldsymbol{w}^N \circ \boldsymbol{w}^{N+1} \tag{128}$$

where $\boldsymbol{w}^n \in R^{I_n}, n = 1, \cdots, N, \boldsymbol{w}^{N+1} \in R^J$. Then the number of elements is reduced from the original $J \prod_{n=1}^{N} I_n$ to $J + \sum_{n=1}^{N} I_n$.

Finally, we introduce a **latent conditional high-order Boltzmann machines(CHBM)**. (Huang *et al.*) [151] proposed latent conditional high-order Boltzmann machine for classification. The algorithm is similar to the high-order Boltzmann machine of the three sets of variables we just mentioned. However, in CHBM, input data are two N sample features $\boldsymbol{x}^i \in R^I, \boldsymbol{y}^i \in R^J, i = 1, \cdots, N$ and z is the relationship label of $\boldsymbol{x}^i, \boldsymbol{y}^i$ where $z = [z_1, z_2]$. For each sample, if $\boldsymbol{x}$ and $\boldsymbol{y}$ are matched, $z = [1, 0]$, else $z = [0, 1]$ ("one-hot" encoding). Then the author adds another set of binary-valued latent variables to the hidden layer. The entire structure is shown in figure 40. Where $\boldsymbol{h}$ denotes the intrinsic relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$. $\boldsymbol{h}$ and z are connected by a weight matrix U. Then its energy function is as follows:

$$\begin{aligned} E(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h}, z) = \underline{W} \times_{1v} \boldsymbol{x} \times_{2v} \boldsymbol{y} \times_{3v} \boldsymbol{h} - \boldsymbol{h}^T U z \\ - \boldsymbol{a}^T \boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{y} - \boldsymbol{c}^T \boldsymbol{h} - \boldsymbol{d}^T z \end{aligned} \tag{129}$$

where $\boldsymbol{a}, \boldsymbol{b}, , \boldsymbol{c}, \boldsymbol{d}$ are the biases of $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{h}$, z, respectively.

Then the value of $z_t, t = \{1, 2\}$ (which is also known as activation conditional probability) is :

$$\begin{aligned} h_k &= p(h_k|x, y) = \sigma(\sum_{ij}^{IJ} w_{ijk} x_i y_j + c_k) \\ z_t &= p(z_t|x, y, h) = \sigma(d_t + \sum_{k}^{K} h_k U_{kt}) \\ & \qquad k = 1, \cdots, K. \quad t = \{1, 2\} \end{aligned} \tag{130}$$

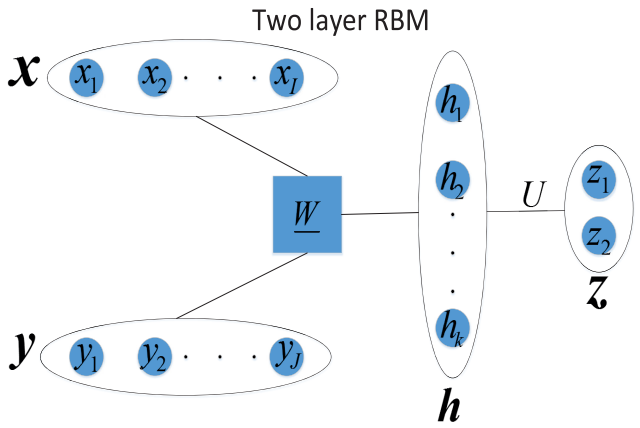

FIGURE 40. Schematic diagram of the energy function of the four sets of variables. The middle is the weight tensor, the above is the variable x, the lower is the variable y, the right is the hidden layer variable h, and the far right is the label z. z and h are connected by a weight matrix U.

In fact, the model is a two-layer RBM. The first layer is ternary RBM $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h})$, and the second layer is the traditional binary RBM $(\boldsymbol{h}, \boldsymbol{z})$. For the 3rd-order tensor $\underline{W}$ of the first layer, we can use the CP decomposition to solve.

3) POLYNOMIAL CLASSIFIER ALGORITHM BASED ON TENSOR TT DECOMPOSITION

Polynomial classifiers are often used for classification because of their ability to generate complex surfaces and have a good fit to raw data. However, when coming to high-dimensional space, the multivariate polynomial can only use some specific kernels in the support vector machine, and the kernel function should be mapped to the high-dimensional space for processing, which increases the difficulty of data processing. In order to enable the Polynomial classifier to handle high dimensional problems, (Chen *et al.*) [161] simplified the operation by using the polynomial with the tensor inner product of the TT format, and proposed two algorithms.

First we give the definition of pure-power-$\boldsymbol{n}$ polynomial: Given a vector $\boldsymbol{n} = (n_1, n_2, \cdots, n_m)$, if in a polynomial f with m variables, the highest power for each variable x_i is n_i, $i = 1, 2, \cdots, m$, then the polynomial f is called pure-power-$\boldsymbol{n}$ polynomial.

Example 1: The polynomial $f = 1 + x_1 + 3x_2^3 + 2x_3 + 4x_3^2 - 2x_2x_3^2 - 5x_1x_2x_3$ is a pure-power-$\boldsymbol{n}$ polynomial with $\boldsymbol{n} = (1, 3, 2)$.

For pure-power-$\boldsymbol{n}$ polynomial, it can be expressed equivalently by the expression of the mode-n product of the vectors and a tensor $\underline{A} \in R^{(n_1+1)\times(n_2+1)\cdots(n_m+1)}$:

$$f = \underline{A} \times_{1v} \boldsymbol{v}(x_1)^T \times_{2v} \boldsymbol{v}(x_2)^T \cdots \times_{mv} \boldsymbol{v}(x_m)^T \quad (131)$$

where $\boldsymbol{v}(x_i)$ are the Vandermonde vectors:

$$\boldsymbol{v}(x_i) = (1, x_i, x_i^2, \cdots, x_i^{n_i})^T, \quad i = 1, 2, \cdots, m \quad (132)$$

Example 2: For the polynomial f in example 1, since n=(1,3,2), then $\boldsymbol{v}(x_1) = (1, x_1)^T$, $\boldsymbol{v}(x_2) = (1, x_2, x_2^2, x_2^3)^T$, and $\boldsymbol{v}(x_3) = (1, x_3, x_3^2)^T$. The nonzero elements of the coefficient tensor $A \in R^{2\times4\times3}$ are $a_{111} = 1$, $a_{211} = 1$, $a_{141} = 3$, $a_{112} = 2$, $a_{113} = 4$, $a_{123} = -2$, $a_{222} = -5$. We combine the indices of the three Vandermonde vectors to get the indices of $\underline{A}$, such as, $-5x_1x_2x_3$ is from $\boldsymbol{v}(x_1)[2] = x_1$, $\boldsymbol{v}(x_2)[2] = x_2$, $\boldsymbol{v}(x_3)[2] = x_3$, so $a_{222} = -5$.

Given a set of N training samples, $(\boldsymbol{x}^i, y^i)$, $i = 1, 2, \cdots, N, \boldsymbol{x}^i \in R^m$. After feature extraction, each feature is mapped to high-dimensional space by mapping T:

$$T(\boldsymbol{x}) = v(x_1) \circ v(x_2) \cdots v(x_m) \quad \boldsymbol{x} = (x_1, x_2, \cdots, x_m)^T \quad (133)$$

Therefore, formula 139 is further equivalent to:

$$f = T(\boldsymbol{x}^i) \bullet \underline{A} \quad (134)$$

Example 3: Here we consider the example of a binary polynomial for the sake of simplicity. Assuming $f = 2+3x_1-x_2+2x_1^2+4x_1x_2-2x_1^2x_2+7x_2^2$. We can get $\boldsymbol{n} = (2, 2)$, $\boldsymbol{v}(x_1) = (1, x_1, x_1^2)^T$, $\boldsymbol{v}(x_2) = (1, x_2, x_2^2)^T$, then according to formula 9 and 17, both $T(\boldsymbol{x})$and A are 2rd-order tensors(matrices):

$$T(\boldsymbol{x}) = \begin{bmatrix} 1 & x_2 & x_2^2 \\ x_1 & x_1x_2 & x_1x_2^2 \\ x_1^2 & x_1^2x_2 & x_1^2x_2^2 \end{bmatrix}, \quad A = \begin{bmatrix} 2 & -1 & 7 \\ 3 & 4 & 0 \\ 2 & -2 & 0 \end{bmatrix} \quad (135)$$

Similar to the idea of SVM, polynomial classification is looking for a hyperplane to distinguish between these two types of examples. Its ultimate goal is to find the coefficient tensor $\underline{A}$ so that:

$$y^i(T(\boldsymbol{x}^i) \bullet \underline{A}) > 0, \quad i = 1, 2 \cdots, N \quad (136)$$

Considering the TT decomposition of the coefficient tensor $\underline{A}$, $\underline{A} = \underline{A}_1 \times_{3,1} \underline{A}_2 \cdots \times_{3,1} \underline{A}_m$, the above polynomial equation (formula 134) has the following further properties:

$$\begin{aligned} f &= T(\boldsymbol{x}^i) \bullet \underline{A} = \underline{A} \times_{1v} \boldsymbol{v}(x_1)^T \times_{2v} \boldsymbol{v}(x_2)^T \cdots \times_{mv} \boldsymbol{v}(x_m)^T \\ &= (\underline{A}_1 \times_{2v} \boldsymbol{v}(x_1)^T) \cdots (\underline{A}_m \times_{2v} \boldsymbol{v}(x_m)^T) \\ &= \underline{A}_j \times_{1v} \boldsymbol{p}_j(\boldsymbol{x}) \times_{2v} \boldsymbol{v}(x_j)^T \times_{3v} \boldsymbol{q}_j(\boldsymbol{x})^T \\ &= (\boldsymbol{q}_j(\boldsymbol{x})^T \otimes_L \boldsymbol{v}(x_j)^T \otimes_L \boldsymbol{p}_j(\boldsymbol{x}))(\underline{A}_j)_{v1} \\ &\quad \textit{for any } j = 1, 2 \cdots, m \end{aligned} \quad (137)$$

where $\boldsymbol{p}_1(\boldsymbol{x}) = 1$, $\boldsymbol{p}_j(\boldsymbol{x})_{j\geq2} = \prod_{k=1}^{j-1}(\underline{A}_k \times_{2v} \boldsymbol{v}(x_k)^T)$ and $\boldsymbol{q}_m(\boldsymbol{x}) = 1$, $\boldsymbol{q}_j(\boldsymbol{x})_{j<m} = \prod_{k=j+1}^{m}(\underline{A}_k \times_{2v} \boldsymbol{v}(x_k)^T)$, $vec(\underline{A}_i)_2$ means the mode-2 vectorization of the tensor (see formula 15).

Example 4: For the polynomial f in example 3, according to formula 137, $T(\boldsymbol{x}) \bullet \underline{A} = (\boldsymbol{q}_2(\boldsymbol{x})^T \otimes_L \boldsymbol{v}(x_2)^T \otimes_L \boldsymbol{p}_2(\boldsymbol{x}))vec(\underline{A}_2)_2$, let $i = 2$. Then we will get:

$$\begin{aligned} \boldsymbol{q}_2(\boldsymbol{x}) &= 1 \\ \boldsymbol{v}(x_2) &= \begin{bmatrix} 1 & x_2 & x_2^2 \end{bmatrix} \\ \boldsymbol{p}_2(\boldsymbol{x})) &= \underline{A}_1 \times_{2v} \boldsymbol{v}(x_1)^T \\ \boldsymbol{v}(x_1) &= \begin{bmatrix} 1 & x_1 & x_1^2 \end{bmatrix} \end{aligned} \quad (138)$$

(Chen *et al.*) [161] proposed two loss functions, least squares loss and logistic loss function:

$$J(\underline{A}) = \frac{1}{N}\sum_{i=1}^{N}(T(\boldsymbol{x}^i) \bullet \underline{A} - y^i)^2$$
$$J(\underline{A}) = -\frac{1}{N}\sum_{i=1}^{N}[\frac{1+y^i}{2} ln(g_{\underline{A}}(\boldsymbol{x}^i)) + \frac{1-y^i}{2} ln(1 - g_{\underline{A}}(\boldsymbol{x}^i))] \tag{139}$$

where the first formula is the least squares loss function, the second is the logical loss function, and $g_{\underline{A}}(\boldsymbol{x}^i) = \sigma(T(\boldsymbol{x}^i) \bullet \underline{A})$, $\sigma(x) = \frac{1}{1+e^{-x}}$.

According to formula 137, the least squares loss function of formula 139 can be further transformed into:

$$J(\underline{A}) = \frac{1}{N}\|C_j(\underline{A}_i)_{v1} - \boldsymbol{y}\|_2^2$$
$$T(\boldsymbol{x}^i) \bullet \underline{A} = C_j[i](\underline{A}_i)_{v1} \tag{140}$$

where

$$C_j = \begin{bmatrix} \boldsymbol{q}_j(\boldsymbol{x}^1)^T \otimes_L \boldsymbol{v}(x_j^1)^T \otimes_L \boldsymbol{p}_j(\boldsymbol{x}^1) \\ \boldsymbol{q}_j(\boldsymbol{x}^2)^T \otimes_L \boldsymbol{v}(x_j^2)^T \otimes_L \boldsymbol{p}_j(\boldsymbol{x}^2) \\ \cdots \\ \boldsymbol{q}_j(\boldsymbol{x}^N)^T \otimes_L \boldsymbol{v}(x_j^N)^T \otimes_L \boldsymbol{p}_j(\boldsymbol{x}^N) \end{bmatrix} \tag{141}$$

and $C_j[i]$ means the ith term of vector $C_j[i]$, $\boldsymbol{y} = [y^1, y^2, \cdots, y^N]^T$.

If we further add a regular term, the final loss function is:

$$J_{loss}(\underline{A}) = J(\underline{A}) + \frac{\alpha}{2}(\underline{A} \bullet \underline{A}) \tag{142}$$

Finally, it is transformed into solving the loss function problem of minimizing the tensor A with TT format. In fact, the idea of optimization is still similar to alternating least squares, which we call the improved alternating least squares method. The central idea is to update only one core A_n in each iteration while keeping other cores unchanged. In general, we first update from A_1 to A_n, so that the left half is updated, which we call forward half-sweep. Then we update from A_n to A_1, and the right half is updated, which we call backward half-sweep. When both forward and backward are completed, an iteration is completed (see figure 41 and algorithm 13).

Algorithm 13 The Improved Least Squares Method for TT Decomposition (Chen *et al.*) [161]

Input:
Loss function $J_{loss}(\underline{A})$ and an initial guess for the TT decomposition of the Nth-order tensor $\underline{A} = \underline{A}_1 \times_{3,1} \underline{A}_2 \cdots \times_{3,1} \underline{A}_m$, $\underline{A}_n \in R^{R_{n-1} \times I_n \times R_n}$;
Output:
A in the TT format, $\underline{A} = argmin\ J_{loss}(\underline{A})$, $\underline{A} = \widehat{\underline{A}}_1 \times_{3,1} \widehat{\underline{A}}_2 \cdots \times_{3,1} \widehat{\underline{A}}_m$;
if the required number of iterations is not reached **then**
  for n=1 to N **do**
    solve: $\widetilde{\underline{A}}^n = \arg\min_{\widetilde{\underline{A}}^n} J(\widetilde{\underline{A}}^1, \cdots, \widetilde{\underline{A}}^{n-1}, \widetilde{\underline{A}}^n, \underline{A}^{n+1}, \cdots, \underline{A}^N)$;
    $[Q^n, R^n] = QR - decomposition\ of\ \widetilde{\underline{A}}^n_{mc2}$;
    $\widetilde{\underline{A}}^{n+1} = \widetilde{\underline{A}}^{n+1} \times_{2m} R^n$;
  end for
  for n=N to 1 **do**
    solve: $\widehat{\underline{A}}^n = \arg\min_{\widehat{\underline{A}}^n} J(\widehat{\underline{A}}^1, \cdots, \widehat{\underline{A}}^{n-1}, \widehat{\underline{A}}^n, \widetilde{\underline{A}}^{n+1}, \cdots, \widetilde{\underline{A}}^N)$;
    $[Q^n, R^n] = QR - decomposition\ of\ \widehat{\underline{A}}^n_{mc2}$;
    $\widehat{\underline{A}}^{n-1} = \widehat{\underline{A}}^{n-1} \times_{2m} R^n$;
  end for
end if
return $\underline{A} = \widehat{\underline{A}}_1 \times_{3,1} \widehat{\underline{A}}_2 \cdots \times_{3,1} \widehat{\underline{A}}_m$;

4) FEATURE TENSOR GENERATION (TENSORIZATION) FOR IMAGE CLASSIFICATION

Feature tensor generation (Tensorization) is often used in image processing, which means finding good image feature representations. By finding feature tensors, that is, extracting valid data, image classification can be better, thereby improving classification accuracy. Usually we need to use some means to convert 2D images into 3D feature tensors to extract information.

Feature tensor generation transforms the original image X into another 3rd-order high-dimensional image $\underline{Y}$, which can maintain the spatial relationship between the images. The size of each transformed image $\underline{Y}$ is much smaller than the original image X, and the original image X can be accurately recovered from the transformed 3D image $\underline{Y}$.

Image-based feature tensor generation is generally generated by the following steps (see algorithm 14).

We also made a picture to show the process of generating feature tensors (see figure 42)

We can recover the original image by reversing the above steps. For the feature tensor, it is highly compatible with the deep learning method commonly used in images, Convolutional Neural Network (CNN). So for general image processing, it can be classified firstly by finding the feature tensor of the image and then we can use CNN to classify. Similar to CNN's convolutional layer, this operation reduces the size of the original image because n and k are smaller than the size N of the original input image, which can significantly reduce computing time and memory consumption. For details, please refer to (Yang *et al.*) [51].

5) TENSOR-BASED FEATURE FUSION FOR FACE RECOGNITION

In general, traditional face recognition has only a single input $\boldsymbol{x}$, and the output expression is as follows:

$$\boldsymbol{y} = f(W^T\boldsymbol{x} + b) \tag{143}$$

But (Hu *et al.*) [37] proposed to combine the face attribute feature and the face recognition feature, which is simply adding an input $\boldsymbol{z}$. Then their output model becomes as

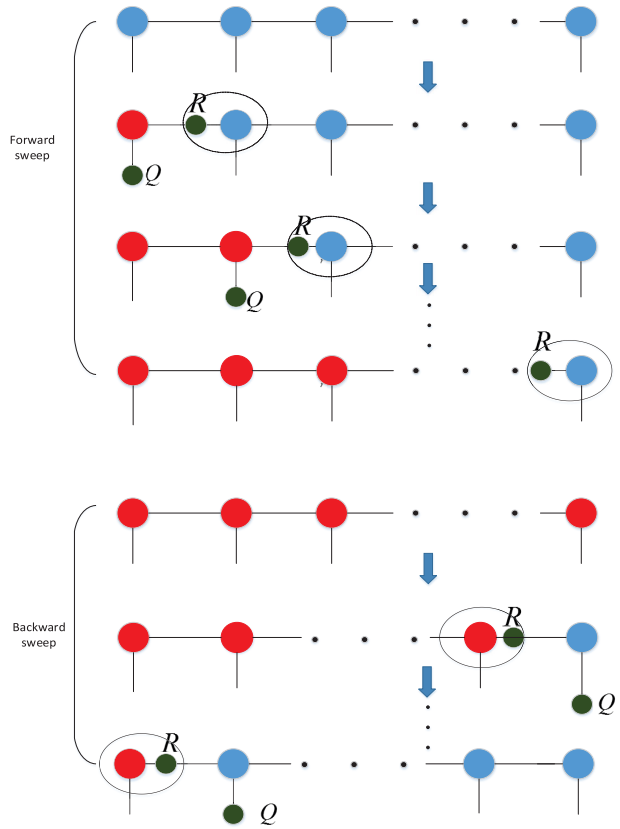

FIGURE 41. The improved least squares method for TT decomposition [161]. First forward half sweep, then backward half sweep, forward and backward half sweep is an iteration. The forward and backward half sweeps are all completed, then an iterative update is completed. After each update of the nuclear tensor, the green matrix R generated by QR decomposition of the nuclear tensor is absorbed into the adjacent matrix, and then continues to update the adjacent matrix.

follows:

$$\begin{aligned} \boldsymbol{y}^i &= softmax(\underline{W} \times_{1v} \boldsymbol{x}^i \times_{3v} \boldsymbol{z}^i) \\ softmax(x_i) &= \frac{e^i}{\sum\limits_j e^j} \end{aligned} \tag{144}$$

where the bias is omitted, $\boldsymbol{x}^i \in R^A$, $\boldsymbol{z}^i \in R^B$, $\boldsymbol{y}^i \in R^C$, $i = 1, \cdots, N$ and weight tensor $\underline{W} \in R^{A \times C \times B}$. The goal is still to optimize the loss function between the predicted and true values. The author used Tucker decomposition

Algorithm 14 Feature Tensor Generation (Yang *et al.*) [51]

Input:

The original image $X \in R^{N \times N}$;

Output:

Feature tensor, which we call $\underline{F}_t$;

1: First, the original input image X is divided into $n \times n$ block regions, and then multi-level perception is performed to find the feature representation of each block region, we call the block area $Y_{a,b}(a, b = 1, 2, \cdots, n)$;

2: Perform DCT transform for each block: $D_{a,b}(u, v) = c(u)c(v) \sum\limits_{x=0}^{K} \sum\limits_{y=0}^{K} Y_{a,b}(x, y) cos[\frac{\pi}{K}(x + \frac{1}{2})u] cos[\frac{\pi}{K}(y + \frac{1}{2})v]$, $K = \frac{N}{n}$, if u=0, $c(u) = \sqrt{\frac{1}{K+1}}$, else, $c(u) = \sqrt{\frac{2}{K+1}}$;

3: Convert matrices $D_{a,b}(u, v)$ to vectors (vectorization), $\boldsymbol{C}_{a,b} = [D_{a,b}(0, 0), D_{a,b}(0, 1), D_{a,b}(0, 2), \cdots, D_{a,b}(K, K)]$;

4: Pick first k elements of each $\boldsymbol{C}_{a,b}$, $\boldsymbol{C}_{ab} = \boldsymbol{C}_{a,b}[0 : k]$;

5: Finally, all these element groups will become a feature tensor $\underline{F}_k = \begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} & \cdots & \boldsymbol{C}_{1n} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} & \cdots & \boldsymbol{C}_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{C}_{n1} & \boldsymbol{C}_{n2} & \cdots & \boldsymbol{C}_{nn} \end{bmatrix}$, where $\boldsymbol{C}_{ab} \in R^k$ are vectors, $\underline{F}_k \in R^{n \times n \times t}$ is a 3rd-order tensor;

$\underline{W} = \underline{S} \times_{1m} U_A \times_{2m} U_C \times_{3m} U_B$, $U_A \in R^{A \times R_A}$, $U_B \in R^{B \times R_B}$, $\underline{S} \in R^{R_A \times R_C \times R_B}$. Then formula 144 becomes as follows:

$$\boldsymbol{y}^i = softmax(\underline{S} \times_{1m} U_A \times_{2m} U_C \times_{3m} U_B \times_{1v} \boldsymbol{x}^i \times_{3v} \boldsymbol{z}^i) \tag{145}$$

According to some properties between the formula 169, it can be converted into:

$$\boldsymbol{y}^i = softmax(\underline{S} \times_{1m} (U_A \boldsymbol{x}^i) \times_{2m} U_C \times_{3m} (U_B \boldsymbol{z}^i)) \tag{146}$$

According to the nature of Kronecker, it can be further transformed into:

$$\boldsymbol{y}^i = softmax(((U_A \boldsymbol{x}^i) \otimes (U_B \boldsymbol{z}^i)) mat(S)_2^T U_C) \tag{147}$$

where $((U_A \boldsymbol{x}^i) \otimes (U_B \boldsymbol{z}^i)) mat(S)_2^T$ is called fused feature. The entire classification process is shown in figure 43.

Finally, the entire training process is actually the process of solving the factor matrix and the core tensor. This way

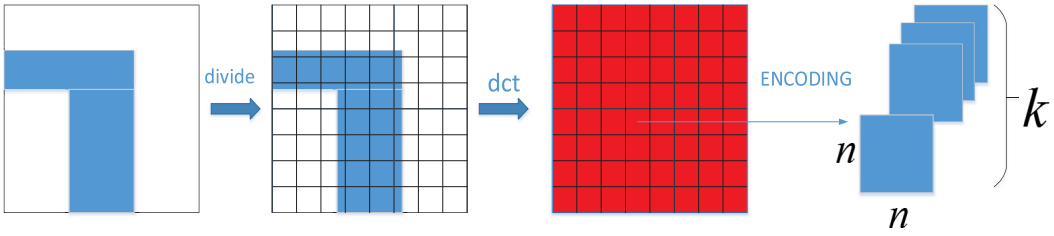

FIGURE 42. Feature tensor generation example (n=8). We assume that the original image (800 × 800) is divided into 8 × 8 blocks, and each block is 100 × 100. Then we perform DCT transformation. Finally it is encoded into a feature tensor 8 × 8 × 100.

of decomposing can reduce the amount of parameters, thus reducing the computation time, and the efficiency of large-scale data processing can be improved.

6) TENSOR-BASED GRAPH EMBEDDING ALGORITHM

The graph embedding algorithm is generally used to better classify data by reducing the dimensionality of the data while preserving the data structure of the graph. In order to accurately classify and identify the target object in the image, (Hu *et al.*) [143] used the second-order tensor-based graph embedding to learn the discriminant subspace (discriminate the embedding space), and distinguished the target object image block and the background image block from the discriminant subspace.

First they assume that input training sample set are Nth-order tensors, $\underline{X}_a \in R^{R_1 \times R_2 \cdots \times R_n}, a = 1, 2 \cdots, N$. They construct an intrinsic graph $\mathcal{G}^i$ to characterize the correlation between the target sample and the background sample. In addition they also construct a penalty graph $\mathcal{G}^p$ to characterize the difference between the target sample and the background sample to separate them from the image. These two graphs represent the geometry and discriminant structure of the input sample. Define the weight matrix of the two graphs separately, W^i, W^p. The element W^i_{ab} in W^i represents the degree of similarity between the vertices X_a and X_b, and the element W^p_{ab} in W^p represents the degree of difference between X_a and X_b.

Tensor-based graph embedding aims to find a best low-dimensional tensor representation for each vertex in a graph $\mathcal{G}$, and to make the low-dimensional tensor well describe the similarity between vertices. The optimal tensor representation of the vertex is obtained by solving the following optimization problem.

$$\begin{aligned} &J(B_1, B_2, \cdots, B_N) \\ &= \arg\min_{B_n} \Big(\sum_{a=1}^{N} \sum_{b=1}^{N} \| \underline{X}_a \times_{1m} B_1 \\ &\quad \cdots \times_{Nm} B_N - \underline{X}_b \times_{1m} B_1 \cdots \times_{Nm} B_N \|_F^2 W^i_{ab} \Big) \\ &\quad s.t. \quad \sum_{a=1}^{N} \sum_{b=1}^{N} \| \underline{X}_a \times_{1m} B_1 \cdots \times_{Nm} B_N \\ &\qquad - \underline{X}_b \times_{1m} B_1 \cdots \times_{Nm} B_N \|_F^2 W^p_{ab} = d \end{aligned} \tag{148}$$

where $B_n \in R^{I_n \times R_n}$ are called transfer matrices, d is a constant according to the needs. In fact, we can see that this is similar to the optimal solution to the Tucker(HOSVD) decomposition with constraints. B_n is actually factor matrices, and $\underline{X}$ is the core tensor in Tucker decomposition. But note that here $I_n \leq R_n$.

However, depending on the image itself, it can be seen as a matrix. (He *et al.*) [146]proposed the solution to the above problem. First, the mode-n matricization of the tensor is used to convert the above optimization problem equivalently. Note that since the input $\underline{X}_a \in R^{R_1 \times R_2}, a = 1, 2 \cdots, N$ are second-order tensors, according to the definition of the mode-n matricization, Then the following formula is established:

$$\underline{X}_{m1} = X, \underline{X}_{m2} = X^T \tag{149}$$

According to figure 20, the mode-n product of the tensor and matrix can be converted into matrix product. Then the $\underline{Y}_a = \underline{X}_a \times_{1m} B_1 \times_{2m} B_2$ in the formula 148 becomes as follow:

$$\begin{aligned} \underline{Y}_a &= \underline{X}_a \times_{1m} B_1 \times_{2m} B_2, \\ \underline{A}^1 &= \underline{A}^1_{m1} = B_1 (\underline{X}_a)_{m1} = B_1 \underline{X}_a, \\ \underline{Y}_a &= \underline{A}^1 \times_{2m} B_2, \\ (\underline{Y}_a)_{m2} &= B_2 (\underline{A}^1)_{m2}, \\ \underline{Y}_a &= ((\underline{Y}_a)_{m2})^T = (B_2 (\underline{A}^1)_{m2})^T = B_1 \underline{X}_a B_2^T \end{aligned} \tag{150}$$

The optimization problem of factor formula 148 is converted into:

$$\begin{aligned} J(B_1, B_2) &= \arg\min_{B_1, B_2} \Big(\sum_{a=1}^{N} \sum_{b=1}^{N} \| B_1 \underline{X}_a B_2^T - B_1 \underline{X}_a B_2^T \|_F^2 W^i_{ab} \Big) \\ &\quad s.t. \quad \sum_{a=1}^{N} \sum_{b=1}^{N} \| B_1 \underline{X}_a B_2^T - B_1 \underline{X}_b B_2^T \|_F^2 W^p_{ab} = d \end{aligned} \tag{151}$$

To further simplify the operation, (He *et al.*) [146] defined two diagonal matrices $\Lambda^i \in R^{N \times N}$ and $\Lambda^p \in R^{N \times N}$, where the diagonal elements are $\lambda^i_{aa} = \sum_{b=1}^{N} W^i_{ab}$ and $\lambda^p_{aa} = \sum_{b=1}^{N} W^p_{ab}$, respectively. Formula 151 can be further transformed into:

$$\begin{aligned} J(A, B) &= \arg\min_{A,B} \ trace(B^T (\Lambda^i_A - W^i_A) B) \\ &\quad s.t. \quad trace(B^T (\Lambda^p_A - W^p_A) B) = \frac{d}{2} \end{aligned} \tag{152}$$

For convenience, $A^T = B_1, B^T = B_2$,

$$\begin{aligned} \Lambda^i_A &= \sum_{a=1}^{N} \lambda^i_{aa} \underline{X}_a^T A A^T \underline{X}_a, \quad W^i_A = \sum_{a=1}^{N} \sum_{b=1}^{N} W^i_{ab} \underline{X}_a^T A A^T \underline{X}_a \\ \Lambda^p_A &= \sum_{a=1}^{N} \lambda^p_{aa} \underline{X}_a^T A A^T \underline{X}_a, \quad W^p_A = \sum_{a=1}^{N} \sum_{b=1}^{N} W^p_{ab} \underline{X}_a^T A A^T \underline{X}_a \end{aligned} \tag{153}$$

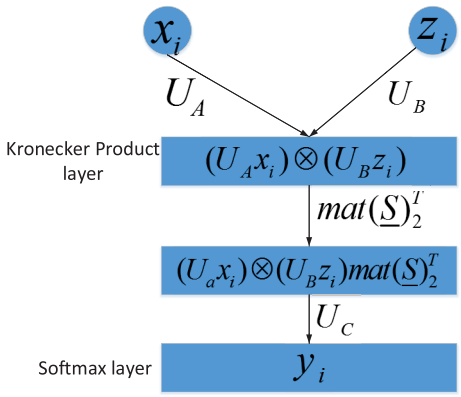

FIGURE 43. Tensor-based feature fusion neural network.

Using the idea of alternating least squares, when fixing A, B consists of I_1 generalized eigenvectors, which correspond to the largest eigenvalues of first I_1 and satisfy the following equation, $(\Lambda_A^p - W_A^p)b = c(\Lambda_A^i - W_A^i)b$. when fixing B, A consists of I_2 generalized eigenvectors, which correspond to the largest eigenvalues of first I_2 and satisfy the following equation, $(\Lambda_B^p - W_B^p)a = c(\Lambda_B^i - W_B^i)a$.

According to the above analysis, we present a graph embedding algorithm based on a second-order tensor (matrix) (see algorithm 15).

Algorithm 15 2nd-Order Tensor-Based Graph Embedding Algorithm (Hu *et al.*) [143]

Input:

Input tensor sample set $\underline{X}_a \in R^{R_1 \times R_2}, a = 1, 2 \cdots, N$;

Output:

Transfer matrices (factor matrices) A^T, B^T;

1: Initially A, take the first I_1 column of the unit matrix $I \in R^{R_1 \times R_1}$ as the matrix A;
2: Initialize the weight coefficient W_{ab}^i and W_{ab}^p according to (Weiming Hu,2017);
3: **for** k=1 to n **do**
4: Calculate Λ_A^i, W_A^i, Λ_A^p, W_A^p in formula 153;
5: Calculate B by using the properties of generalized eigenvectors: $(\Lambda_A^p - W_A^p)b = c(\Lambda_A^i - W_A^i)b$;
6: Calculate Λ_B^i, W_B^i, Λ_B^p, W_B^p in formula 153 by exchanging A and B;
7: Replace A by using the properties of generalized eigenvectors: $(\Lambda_B^p - W_B^p)a = c(\Lambda_B^i - W_B^i)a$;
8: **end for**
9: **return** Transfer matrices (factor matrices) A^T, B^T;

Note that since the images are mostly two-dimensional, the above authors used the form of a second-order matrix. In fact, if the input is a higher-dimension tensor (n>3), we can still use the alternating least squares to convert the above multivariate optimization problem into a single variable optimization problem.

7) DISCUSSION AND COMPARISON

This section mainly introduces some tensor-based classification algorithms. Among them, Support Tensor Machine (STM) and High-order Restricted Boltzmann machines (HORBM) are the main ones. Similar to the traditional SVM, STM can only perform two classifications. When faced with multiple classification problems, we need to perform STM multiple times, and now the more popular method is to use neural network replacement. Due to the simplicity of rank-one decomposition, we mainly describe the STM algorithm based on rank-one decomposition in STM. For high-order Boltzmann machines, we extend it to a more general case. At the same time, in the case of performing CP decomposition or rank-one decomposition of tensor, not only the number of unknown parameters is reduced, but also efficient iterative solution can be effectively performed by ALS. Tensor can also combine various high-dimensional features to improve the classification accuracy. Therefore, a tensor-based feature fusion technique is proposed for classification processing. Finally, using the tensor to separate the target from the background pattern, the discriminant space can be effectively learned, thereby effectively detecting the target in the picture.

C. APPLICATION OF TENSOR IN DATA PREPROCESSING

1) TENSOR DICTIONARY LEARNING

Dictionary learning refers to finding a sparse representation of the original data while ensuring the structure and non-distortion of the data, thereby achieving the effect of data compression and ultimately reducing computational complexity (see figure 44). General dictionary learning boils down to the following optimization problems:

$$\min_{A,x_i} \sum_{i=1}^{N} ||\boldsymbol{y_i} - A\boldsymbol{x_i}||_2^2 + \lambda \sum_{i=1}^{N} ||\boldsymbol{x_i}||_1, \quad i = 1, \cdots, N \tag{154}$$

where $A \in R^{J \times I}$ is sparse matrix, $\boldsymbol{y}_i \in R^J$, $i = 1, \cdots, N$ are N raw data and $\boldsymbol{x}_i \in R^I$ are vectors sparsely represented.

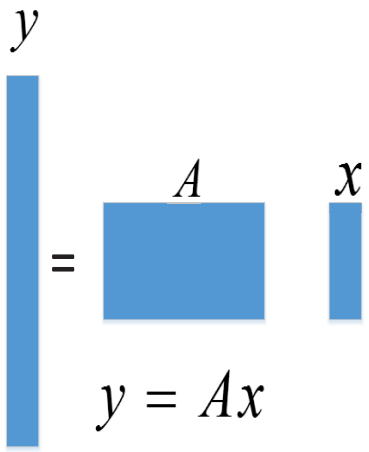

FIGURE 44. Schematic diagram of dictionary learning, left y is the original input data, A is a sparse matrix, and x is a sparse representation of y.

We now extend the vector to the tensor. When the input raw data $\underline{Y} \in R^{I_1 \times I_2 \times \cdots I_N}$ is an Nth-order tensor, it produces tensor dictionary learning. The tensor decomposition method is usually used to solve the problem for the tensor dictionary learning. (Ghassemi *et al.*) [88] used the Kronecker product representation of Tucker decomposition to represent the above optimization problem. According to the expression of the formula 108 Tucker decomposition, we get

$$\underline{Y}_{v1} = (B_N \otimes_R B_{N-1} \cdots \otimes_R B_1)\underline{X}_{v1} \tag{155}$$

We combine N samples as N column vectors of a new matrix Y, and get the expression of the matrix as follows:

$$Y = (B_N \otimes_R B_{N-1} \cdots \otimes_R B_1)X \tag{156}$$

At this time we call the factor matrices the Kronecker structured(KS) matrices and let $D = B_N \otimes_L B_{N-1} \cdots \otimes_L B_1$.

However, a more general sparse matrix is a low rank separation structure matrix, which is the sum of KS matrices, as follows:

$$D = \sum_{i=1}^{I} B_N^i \otimes_R B_{N-1}^i \cdots \otimes_R B_1^i \tag{157}$$

Considering another property. Let $D = B_2 \otimes_R B_1$, for the elements in D we can reconstitute the form of the vector outer product as follows: $D^r = vec(B_1)_1 \circ vec(B_2)_1$. Then we can convert the equivalent of equation 157 to the following:

$$D^r = \sum_{i=1}^{I} (B_1^i)_{v1} \circ (B_2^i)_{v1} \cdots \circ (B_N^i)_{v1} \tag{158}$$

So we can use this structure as a regular term. Finally, we get the optimal expression for the tensor dictionary learning as follows:

$$\min_{D,X} \frac{1}{2}||Y - DX||_F^2 + \lambda \frac{1}{N} \sum_{n=1}^{N} ||D_{mn}^r||_* \tag{159}$$

where $D = \sum_{i=1}^{I} B_N^i \otimes_R B_{N-1}^i \cdots \otimes_R B_1^i$, $||D_{mn}^r||_*$ is the kernel norm of the matrix D^r after the mode-n matricization of the tensor $\underline{D}^r$. It is generally solved by the Lagrangian multiplier method. Since the solution process is too complicated, it is omitted here. For details, please refer to (Ghassemi *et al.*) [88].

2) TENSOR COMPLETION FOR DATA PROCESSING

In data processing, sometimes there are some missing values in the data. There are many ways to complete the missing data, and the popular ones are matrix estimation and matrix completion. If the input data is tensor, then we call the tensor estimation and tensor completion. The tensor estimation and the tensor completion are similar. They are all required to solve the corresponding minimum constraint problem. However, the tensor estimation is mainly to minimize the mean square error between the estimated value and the original value. Here we mainly introduce the tensor completion. The general tensor completion aims to seek the optimal solution of the following expression:

$$\min_{\underline{Y}} \ \|(\underline{X} - \underline{Y}) \circledast \widetilde{\underline{I}}\|_F^2 \tag{160}$$

where $\underline{X} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ is a tensor with missing values, $\underline{Y} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ means the reconstruction tensor, $\circledast$ means the element product (see formula 19), and $\widetilde{\underline{I}} \in R^{I_1 \times I_2 \times \cdots \times I_N}$ represents the indexes of missing values in $\underline{X}$. The entries of $\widetilde{\underline{I}}$ are as follows:

$$\widetilde{\underline{i}}_{i_1 i_2 \cdots i_N} == \begin{cases} 0 & if\ \underline{x}_{i_1 i_2 \cdots i_N}\ is\ missing \\ 1 & otherwise \end{cases} \tag{161}$$

The first step in such problems is usually to find a low rank approximation of the original tensor. The conventional method uses several tensor decompositions introduced in part one, such as CP, HOSVD, TT decomposition, etc. (Peng *et al.*) [68] used the HOSVD decomposition. But he did not use the traditional truncated SVD decomposition algorithm (see algorithm 2). Since traditional algorithms need to initialize the approximate rank of a given tensor first and the factor matrices, which actually requires a lot of precalculation. So they proposed an adaptive algorithm to obtain the low rank approximation of tensors.

First they set an error parameter $\alpha \in [0, 1]$. Then, similar to truncated-SVD, SVD decomposition is performed on the mode-n matricization of the core tensor $\underline{\Lambda}_{mk}$, $k = 1, 2 \cdots, N$, $\underline{\Lambda}_{mk} = U_k S_k V^T$. Where S is a diagonal matrix with nonzero entries $s_{jj}, j = 1, \cdots, K$, $K = rank(\underline{\Lambda}_{mk})$. The optimal rank can be obtained by

$$R_k = \min_{R_k}(R_0 < R_k < I_k < \frac{\sum_{j=R_k+1}^{K} s_{jj}}{\sum_{j=1}^{K} s_{jj}} < \alpha) \tag{162}$$

where R_0 is the lower bound of the predefined rank, which prevents the rank from being too small. The detailed process is shown in algorithm 16.

Algorithm 16 The Adaptive HOSVD Decomposition of the Tensor (Peng *et al.*) [68]

Input:
The Nth-order data tensor $\underline{X} \in R^{I_1 \times I_2 \cdots I_N}$, error parameter $\alpha \in [0, 1]$, R_0

Output:
The core tensor $\underline{A} \in R^{R_1 \times R_2 \cdots R_N}$ and the factor matrices $B_n \in R^{I_n \times R_n}$;

1: $\underline{A}^0 \leftarrow \underline{X}$;
2: **for** n=1 to N **do**
3: $[U_n, S_n, V_n^T] = SVD(\underline{A}_{mn}^{n-1})$, and then compute rank R_k by formula 90;
4: Select the first R_k column vector of U to assign to the factor matrix B_k, $B_k = U_n(:, 1 : R_k) = [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_{R_k}]$;
5: $\underline{A}_{mn}^n = S_n(1 : R_K, 1 : R_K)V_n(:, 1 : R_k)^T$;
6: **end for**
7: $\underline{A} = \underline{A}^N$;
8: **return** the core tensor $\underline{A}$ and factor matrices B_n

When the improved HOSVD decomposition algorithm is completed, we obtain the factor matrices B_n and the low rank approximate solution of the original tensor $\underline{X}$ by the mode-n product of the original input tensor and the factor matrices, as follows:

$$\underline{Z}_i = \underline{X} \times_{1m} (B_1 B_1^T) \times_{2m} (B_2 B_2^T) \cdots \times_{im} (B_i B_i^T) \tag{163}$$

where $i = 1, \cdots, N$, so we can get N low rank approximate solutions of $\underline{X}$: $\underline{Z}_1, \underline{Z}_2, \cdots, \underline{Z}_N$. We take the average of these N numbers as a best approximation of the original tensor $\underline{X}$. $\underline{X} \approx \frac{1}{N} \sum_{i=1}^{N} \underline{Z}_i$.

After the previous steps, we first perform a zero-compensation operation for the missing data of $\underline{X}$, and we get the fulfilled tensor $\widehat{\underline{X}}$. And then we get the approximate solution of it.

$$D = \frac{1}{N}\sum_{i=1}^{N}\widehat{\underline{Z}}_i \tag{164}$$

Finally, for missing values, we update it with the following formula:

$$\widehat{\underline{X}} = \underline{X} \circledast \widetilde{\underline{I}} + D \circledast (\neg\widetilde{\underline{I}}) \tag{165}$$

where $\neg$ is the Boolean NOT operator (i.e., $0 \leftarrow 1$, $1 \leftarrow 0$). The entire tensor completion algorithm is shown in algorithm 17.

Algorithm 17 Tensor Completion (Zisen Fang, 2018)

Input:

The Nth-order data tensor $\underline{X} \in R^{I_1 \times I_2 \cdots I_N}$ with missing values, $\widetilde{\underline{I}}$, error parameter $\alpha \in [0, 1]$, R_0, the number of iterations required L

Output:

The Nth-order tensor $\widehat{\underline{X}}$ obtained after the missing value is completed, the core tensor $\underline{A} \in R^{R_1 \times R_2 \cdots R_N}$ and the factor matrices $B_n \in R^{I_n \times R_n}$;

$\widehat{\underline{X}}_0 \leftarrow \underline{X} \circledast \widetilde{\underline{I}}$, $D_0 = 0$;
for n=1 to L **do**
  Obtain the core tensor $\underline{A}$ and factor matrices B_n by applying algorithm 16 for $\widehat{\underline{X}}_{n-1}$;
  for k=1 to N **do**
    $\underline{Z}_k = \underline{X} \times_{1m} (B_1 B_1^T) \times_{2m} (B_2 B_2^T) \cdots \times_{km} (B_k B_k^T)$;
  end for
  $D_n = \frac{1}{N}\sum_{k=1}^{N}\widehat{\underline{Z}}_k$;
  $\widehat{\underline{X}}_n = \underline{X} \circledast \widetilde{\underline{I}} + D_n \circledast (\neg\widetilde{\underline{I}})$;
end for
$\widehat{\underline{X}} \leftarrow \widehat{\underline{X}}_L$;

3) DISCUSSION AND COMPARISON

This section focuses on two kinds of common data preprocessing, dimensionality reduction and data completion. For dictionary learning, we introduce a tensor model based on Tucker decomposition, and it can be easily solved due to the nature of the Tucker decomposition. At the same time, we also introduce the latest tensor completion algorithm based on improved HOSVD decomposition for the completion of data missing values.

D. BRIEF SUMMARY FOR PART TWO

Part two introduced the applications of tensor algorithms, including data preprocessing, data classification and data prediction (regression). We can see from part two that in order to solve the high-dimensional problem, more and more researchers have begun to develop tensor-based algorithms. The biggest feature of the tensor algorithm is that it can effectively use the data structure to extract effective information. At the same time, we use tensor decomposition to reduce unknown parameters and the size of the original tensor. Finally, the original problem is transformed into a one-variable optimization problem by alternating least squares algorithm. Tensor-based algorithms not only ensure the interrelationship in the data characteristics, but also improve the accuracy.

IV. CHALLENGES AND PROSPECTS

A. CHALLENGES

As a new technology in recent years, tensor is gradually applied to various fields, such as medicine, biology, computer vision, machine learning, etc. But at the same time it also faces many challenges.

For example, for the existing tensor-based tracking algorithm, they cannot completely detect the intrinsic local geometry and discriminant structure of the image block in tensor form. As a result, they often ignore the influence of the background, which will be interfered by the background area and reduce the accuracy of target tracking.

Regardless of the classification problem of machine learning or deep learning, the tensor decomposition also requires more parameters. In order to improve the accuracy, a large number of samples are needed. Without a better training algorithm, a large number of parameters will cause slow convergence or even no convergence. At the same time, how to obtain a huge amount of data is also a very important issue. Due to the limited sample, researchers often choose to experiment on simulated data. After all, the simulation data is different from the actual data, so the accuracy is not fully guaranteed when applying the actual high-dimensional data.

For the traditional tensor decomposition introduced in part one, such as Tucker decomposition and CP decomposition, they all decompose the input tensor into multiple low-order factors. However, due to some noise in illumination, occlusion or practical applications, they are prone to deviations. Thus the accuracy of the decomposition will decrease, which means that the robustness of these decomposition algorithms is relatively poor.

However, for data processing, some general tensor algorithms often directly decompose the input features into multiple dimensions, which excessively consider the combination of these features with other useless features. So it is a huge challenge to accurately extract useful information in the decomposition and abandon the useless combination of features.

The last big problem is about tensor decomposition algorithms. When it comes to tensor decomposition, it is indispensable to talk about alternating least squares, which alternately obtains the factor of tensor decomposition by iteratively updating the single core each time. However, these algorithms have a common problem, that is, the problem of initialization. In deep learning and machine learning, if the weight initialization is not appropriate, it will cause long

convergence time or even non converge. Therefore, how to effectively initialize tensor rank and the factor matrices is a huge challenge.

B. PROSPECTS

For the above problems, we proposed the following research directions:

1. For target detection and image tracking, can we find a tensor-based algorithm that can capture the features between the background image and the target image?

In many cases, we need to extract the target image which we need from an image. At this time, the dynamic video image will be more difficult. How to better grasp the characteristics of the target and the background and distinguish them will affect the accuracy of target tracking. Therefore, we urgently need to develop such a tensor-based tracking algorithm that can capture the geometric local structural relationship and discriminant relationship between background and target image blocks.

2. How to optimize the learning algorithm or avoid the saddle point?

How to improve the traditional gradient descent algorithm or how to avoid the saddle point becomes an urgent requirement based on tensor deep learning. For deep learning, when the dimension becomes higher, the first problem we think of is the increase of computational complexity and computation time. In general, we usually use tensor decomposition for dimensionality reduction. But some new problems are inevitably generated. A more common problem with gradient descent is that it tends to fall into local minima. As the dimension rises, such problems become more widespread, and we still need to invent some improved algorithms to prevent the network from falling into local minima. Also the saddle point is generated due to the high demension, which becomes a non-convex problem. Therefore, the learning update algorithms need to be improved urgently, otherwise the accuracy cannot be improved.

3. Can the non-convex problem of the weight optimization process be transformed into a convex optimization problem?

As the dimension increases, in general, the objective function will change to a non-convex function, which leads to the non-convex optimization problem. Usually the non-convex optimization problem is difficult to solve, so we always want to find its equivalent convex optimization problem to solve. Can we use effective tensor decomposition or other algorithms to transform non-convex objective functions into convex functions and optimize them?

4. How to reduce the required samples and convergence time while ensuring accuracy?

Some researchers tried to convert original tensor problem into a traditional vector problem, which not only destroys the original data structure, but also greatly increases the number of parameters. The current method for tensor data is through tensor decomposition, which directly converts the data tensor into factor matrices and the core tensor. Some researchers have reduced the parameters by tensor contraction calculations (Kossaifi *et al.*) [64]. However, whether using tensor decomposition or tensor contraction to reduce unknown parameters, the actual unknown parameters that need to be solved is still more than ordinary problems. Therefore, for the required sample data, one is to use simulation data, and the other is to increase the sample parameters that are missing in reality by tensor completion. However, there are accuracy problems in both methods, which also affect the training of the later models and so on.

5. Is it possible to improve low-rank tensor decomposition algorithms?

We have mentioned in the last section that tensor decomposition algorithms face the problem of initializing factor matrices, core tensors and tensor rank. For factor matrices and core tensors, we tend to use the usual random Gaussian variables to initialize the parameters. According to the structural characteristics of tensor or the mode-n matricization and vectorization of tensor, we can add some additional priori information to the factor matrices and core tensor. So can we apply some constraints to the factor matrices and core tensor to better find the properties and effectively initialize it? Or can we improve the alternating least squares algorithm so that it can find the original input tensor characteristics and initialize it automatically? For the tensor rank, we have just introduced a new improved algorithm in part two tensor completion algorithm (see algorithm 17).

V. CONCLUSION

This survey focuses on the basics of tensors, including tensor definitions, tensor operations, tensor decomposition, and low-rank tensor-based algorithms. At the same time, we also describe the application of tensor decomposition in various fields and introduce some applications of tensor algorithms in the field of machine learning and deep learning. Finally, we discuss the opportunities and challenges of tensor.

REFERENCES

[1] A. Bibi and B. Ghanem, "High order tensor formulation for convolutional sparse coding," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 1790–1798.

[2] A. Cichocki, "Tensor decompositions: A new concept in brain data analysis?" 2013, *arXiv:1305.0395*. [Online]. Available: https://arxiv.org/abs/1305.0395

[3] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Found. Trends Mach. Learn.*, vol. 9, nos. 4–5, pp. 249–429, 2016.

[4] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, Mar. 2015.

[5] A. Cichocki, R. Zdunek, A. H. Phan, and S.-I. Amari, *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation*. Chichester, U.K.: Wiley, 2009.

[6] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, and I. Oseledets, "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives foundations and trends?" *Mach. Learn.*, vol. 9, no. 6, pp. 431–673, 2017.

[7] A. Desai, M. Ghashami, and J. M. Phillips, "Improved practical matrix sketching with guarantees," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 7, pp. 1678–1690, Jul. 2016.

[8] A.-H. Phan, A. Cichocki, P. Tichavský, D. Mandic, and K. Matsuoka, "On revealing replicating structures in multiway data: A novel tensor decomposition approach," in *Proc. 10th Int. Conf. LVA/ICA*. Berlin, Germany: Springer, Mar. 2012, pp. 297–305.

[9] A. H. Phan and A. Cichocki, "Extended HALS algorithm for nonnegative Tucker decomposition and its applications for multiway analysis and classification," *Neurocomputing*, vol. 74, no. 11, pp. 1956–1969, 2011.

[10] A.-H. Phan, A. Cichocki, A. Uschmajew, P. Tichavsky, G. Luta, and D. Mandic, "Tensor networks for latent variable analysis. Part I: Algorithms for tensor train decomposition," 2016, *arXiv:1609.09230*. [Online]. Available: https://arxiv.org/abs/1609.09230

[11] A. Hyvärinen, "Independent component analysis: Recent advances," *Philos. Trans. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 371, no. 1984, 2013, Art. no. 20110534.

[12] A. Kolbeinsson, J. Kossaifi, Y. Panagakis, A. Bulat, A. Anandkumar, I. Tzoulaki, and P. Matthews, "Robust deep networks with randomized tensor regression layers," 2019, *arXiv:1902.10758*. [Online]. Available: https://arxiv.org/abs/1902.10758

[13] A. Tjandra, S. Sakti, and S. Nakamura, "Tensor decomposition for compressing recurrent neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Rio de Janeiro, Brazil, Jul. 2018, pp. 1–8.

[14] B. Jiang, F. Yang, and S. Zhang, "Tensor and its Tucker core: The invariance relationships," Jan. 2016, *arXiv:1601.01469*. [Online]. Available: https://arxiv.org/abs/1601.01469

[15] B. Mao, Z. M. Fadlullah, F. Tang, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "A tensor based deep learning technique for intelligent packet routing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Singapore, Dec. 2017, pp. 1–6.

[16] B. Khoromskij and A. Veit, "Efficient computation of highly oscillatory integrals by using QTT tensor approximation," *Comput. Methods Appl. Math.*, vol. 16, no. 1, pp. 145–159, 2016.

[17] C. F. Caiafa and A. Cichocki, "Stable, robust, and super fast reconstruction of tensors using multi-way projections," *IEEE Trans. Signal Process.*, vol. 63, no. 3, pp. 780–793, Feb. 2015.

[18] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[19] C. M. Crainiceanu, B. S. Caffo, S. Luo, V. M. Zipunnikov, and N. M. Punjabi, "Population value decomposition, a framework for the analysis of image populations," *J. Amer. Statist. Assoc.*, vol. 106, no. 495, pp. 775–790, 2011.

[20] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, "Tensor robust principal component analysis with a new tensor nuclear norm," *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published.

[21] C. Peng, L. Zou, and D.-S. Huang, "Discovery of relationships between long non-coding RNAs and genes in human diseases based on tensor completion," *IEEE Access*, vol. 6, pp. 59152–59162, 2018.

[22] C. Tobler, "Low-rank tensor methods for linear systems and eigenvalue problems," M.S. thesis, ETH Zürich, Zürich, Switzerland, 2012.

[23] D. Kressner, M. Steinlechner, and A. Uschmajew, "Low-rank tensor methods with subspace correction for symmetric eigenvalue problems," *SIAM J. Sci. Comput.*, vol. 36, no. 5, pp. A2346–A2368, 2014.

[24] D. Kressner, M. Steinlechner, and B. Vandereycken, "Low-rank tensor completion by Riemannian optimization," *BIT Numer. Math.*, vol. 54, no. 2, pp. 447–468, 2014.

[25] D. Kressner and C. Tobler, "Algorithm 941: Htucker—A MATLAB toolbox for tensors in hierarchical Tucker format," *ACM Trans. Math. Softw.*, vol. 40, no. 3, 2014, Art. no. 22.

[26] D. Kressner and A. Uschmajew, "On low-rank approximability of solutions to high-dimensional operator equations and eigenvalue problems," *Linear Algebra Appl.*, vol. 493, pp. 556–572, Mar. 2016.

[27] D. Tao, X. Li, W. Hu, S. Maybank, and X. Wu, "Supervised tensor learning," in *Proc. 5th IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2005, pp. 8–16.

[28] D. Wang, H. Shen, and Y. Truong, "Efficient dimension reduction for high-dimensional matrix-valued data," *Neurocomputing*, vol. 190, pp. 25–34, May 2016.

[29] E. Corona, A. Rahimian, and D. Zorin, "A tensor-train accelerated solver for integral equations in complex geometries," Nov. 2015, *arXiv:1511.06029*. [Online]. Available: https://arxiv.org/abs/1511.06029

[30] E. M. Stoudenmire and D. J. Schwab, "Supervised learning with quantum-inspired tensor networks," 2016, *arXiv:1605.05775*. [Online]. Available: https://arxiv.org/abs/1605.05775

[31] F. L. Hitchcock, "Multiple invariants and generalized rank of a p-way matrix or tensor," *J. Math. Phys.*, vol. 7, pp. 39–79, Apr. 1928.

[32] F. Verstraete, V. Murg, and J. I. Cirac, "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems," *Adv. Phys.*, vol. 57, no. 2, pp. 143–224, 2008.

[33] G. Ballard, N. Knight, and K. Rouse, "Communication lower bounds for matricized tensor times Khatri-Rao product," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, Vancouver, BC, Canada, May 2018, pp. 557–567.

[34] G. Chabriel, M. Kleinsteuber, E. Moreau, H. Shen, P. Tichavsky, and A. Yeredor, "Joint matrices decompositions and blind source separation: A survey of methods, identification, and applications," *IEEE Signal Process. Mag.*, vol. 31, no. 3, pp. 34–43, May 2014.

[35] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, 2002.

[36] G. Evenbly and G. Vidal, "Algorithms for entanglement renormalization," *Phys. Rev. B, Condens. Matter*, vol. 79, no. 14, 2009, Art. no. 144108.

[37] G. Hu, Y. Hua, Y. Yuan, Z. Zhang, Z. Lu, S. S. Mukherjee, T. M. Hospedales, N. M. Robertson, and Y. Yang, "Attribute-enhanced face recognition with neural tensor fusion networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 3764–3773.

[38] G. Lechuga, L. Le Brusquet, V. Perlbarg, L. Puybasset, D. Galanaud, and A. Tenenhaus, "Discriminant analysis for multiway data," in *Proc. Int. Conf. Partial Least Squares Related Methods*, in Springer Proceedings in Mathematics and Statistics, 2015, pp. 115–126.

[39] T. Goldstein, B. Odonoghue, and S. Setzer, "Fast alternating direction optimization methods," CAM Rep., 2012, pp. 12–35.

[40] G. Vidal, "Efficient classical simulation of slightly entangled quantum computations," *Phys. Rev. Lett.*, vol. 91, no. 14, 2003, Art. no. 147902.

[41] G. Zhou and A. Cichocki, "Fast and unique Tucker decompositions via multiway blind source separation," *Bull. Polish Acad. Sci.*, vol. 60, no. 3, pp. 389–407, 2012.

[42] G. Zhou and A. Cichocki, "Canonical polyadic decomposition based on a single mode blind source separation," *IEEE Signal Process. Lett.*, vol. 19, no. 8, pp. 523–526, Aug. 2012.

[43] G. Zhou, A. Cichocki, Y. Zhang, and D. P. Mandic, "Group component analysis for multiblock data: Common and individual feature extraction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 11, pp. 2426–2439, Nov. 2016.

[44] G. Zhou, Q. Zhao, Y. Zhang, T. Adalı, S. Xie, and A. Cichocki, "Linked component analysis from matrices to high-order tensors: Applications to biomedical data," *Proc. IEEE*, vol. 104, no. 2, pp. 310–331, Feb. 2016.

[45] H. Chen, Q. Ren, and Y. Zhang, "A hierarchical support tensor machine structure for target detection on high-resolution remote sensing images," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Fort Worth, TX, USA, Jul. 2017, pp. 594–597.

[46] H. Fanaee-T and J. Gama, "Tensor-based anomaly detection: An interdisciplinary survey," *Knowl.-Based Syst.*, vol. 98, pp. 130–147, Apr. 2016.

[47] H. Imtia and A. D. Sarwate, "Improved algorithms for differentially private orthogonal tensor decomposition," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Calgary, AB, Canada, Apr. 2018, pp. 2201–2205.

[48] H. Lu, L. Zhang, Z. Cao, W. Wei, K. Xian, C. Shen, and A. van den Hengel, "When unsupervised domain adaptation meets tensor representations," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 599–608.

[49] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, "A survey of multilinear subspace learning for tensor data," *Pattern Recognit.*, vol. 44, no. 7, pp. 1540–1551, 2011.

[50] H. Matsueda, "Analytic optimization of a MERA network and its relevance to quantum integrability and wavelet," 2016, *arXiv:1608.02205*. [Online]. Available: https://arxiv.org/abs/1608.02205

[51] H. Yang, J. Su, Y. Zou, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2017, pp. 1–6.

[52] H. Wang, Q. Wu, L. Shi, Y. Yu, and N. Ahuja, "Out-of-core tensor approximation of multi-dimensional matrices of visual data," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 527–535, 2005.

[53] H. Wang, D. Huang, Y. Wang, and H. Yang, "Facial aging simulation via tensor completion and metric learning," *IET Comput. Vis.*, vol. 11, no. 1, pp. 78–86, Feb. 2017.

[54] H. Zhao, Z. Wei, and H. Yan, "Detection of correlated co-clusters in tensor data based on the slice-wise factorization," in *Proc. Int. Conf. Mach. Learn. (ICMLC)*, Ningbo, China, Jul. 2017, pp. 182–188.

[55] H. Zhou, L. Li, and H. Zhu, "Tensor regression with applications in neuroimaging data analysis," *J. Amer. Stat. Assoc.*, vol. 108, no. 502, pp. 540–552, 2013.
[56] I. Jeon, E. E. Papalexakis, C. Faloutsos, L. Sael, and U. Kang, "Mining billion-scale tensors: Algorithms and discoveries," *VLDB J.*, vol. 25, no. 4, pp. 519–544, 2016.
[57] I. Kisil, G. G. Calvi, A. Cichocki, and D. P. Mandic, "Common and individual feature extraction using tensor decompositions: A remedy for the curse of dimensionality?" in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Calgary, AB, Canada, Apr. 2018, pp. 6299–6303.
[58] I. Kotsia, W. Guo, and I. Patras, "Higher rank support tensor machines for visual recognition," *Pattern Recognit.*, vol. 45, no. 12, pp. 4192–4203, 2012.
[59] I. Kotsia and I. Patras, "Support Tucker machines," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2011, pp. 633–640.
[60] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.
[61] I. V. Oseledets and E. E. Tyrtyshnikov, "Breaking the curse of dimensionality, or how to use SVD in many dimensions," *SIAM J. Sci. Comput.*, vol. 31, no. 5, pp. 3744–3759, 2009.
[62] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Randomized single-view algorithms for low-rank matrix approximation," Tech. Rep., 2016.
[63] J. H. Choi and S. Vishwanathan, "DFacTo: Distributed factorization of tensors," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1296–1304.
[64] J. Kossaifi, A. Khanna, Z. Lipton, T. Furlanello, and A. Anandkumar, "Tensor contraction layers for parsimonious deep nets," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Honolulu, HI, USA, Jul. 2017, pp. 1940–1946.
[65] J. Virta and K. Nordhausen, "Blind source separation for nonstationary tensor-valued time series," in *Proc. IEEE 27th Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Tokyo, Japan, Sep. 2017, pp. 1–6.
[66] K. Batselier and N. Wong, "A constructive arbitrary-degree Kronecker product decomposition of tensors," 2015, *arXiv:1507.08805*. [Online]. Available: https://arxiv.org/abs/1507.08805
[67] K. Batselier, H. Liu, and N. Wong, "A constructive algorithm for decomposing a tensor into a finite sum of orthonormal rank-1 terms," *SIAM J. Matrix Anal. Appl.*, vol. 36, no. 3, pp. 1315–1337, 2015.
[68] K.-Y. Peng, S.-Y. Fu, Y.-P. Liu, and W.-C. Hsu, "Adaptive runtime exploiting sparsity in tensor of deep learning neural network on heterogeneous systems," in *Proc. Int. Conf. Embedded Comput. Syst., Archit., Modeling, Simulation (SAMOS)*, Pythagorion, Greece, Jul. 2017, pp. 105–112.
[69] K. Makantasis, A. D. Doulamis, N. D. Doulamis, and A. Nikitakis, "Tensor-based classification models for hyperspectral data analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 56, no. 12, pp. 6884–6898, Dec. 2018.
[70] K. Makantasis, A. Doulamis, N. Doulamis, A. Nikitakis, and A. Voulodimos, "Tensor-based nonlinear classifier for high-order data analysis," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Calgary, AB, Canada, Apr. 2018, pp. 2221–2225.
[71] K. Naskovska and M. Haardt, "Extension of the semi-algebraic framework for approximate CP decompositions via simultaneous matrix diagonalization to the efficient calculation of coupled CP decompositions," in *Proc. 50th Asilomar Conf. Signals, Syst. Comput.*, Pacific Grove, CA, USA, Nov. 2016, pp. 1728–1732.
[72] T. Levi-Civita, *The Absolute Differential Calculus*. London, U.K.: Blackie and Son, 1927.
[73] L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1253–1278, 2000.
[74] L. De Lathauwer, B. De Moor, and J. Vandewalle, "On the best rank-1 and rank-(R_1,R_2,. . .,R_N) approximation of higher-order tensors," *SIAM J. Matrix Anal. Appl.*, vol. 21, no. 4, pp. 1324–1342, 2000.
[75] L. Geng, X. Nie, S. Niu, Y. Yin, and J. Lin, "Structural compact core tensor dictionary learning for multispec-tral remote sensing image deblurring," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Athens, Greece, Oct. 2018, pp. 2865–2869.
[76] L. Albera, H. Becker, A. Karfoul, R. Gribonval, A. Kachenoura, S. Bensaid, L. Senhadji, A. Hernandez, and I. Merlet, "Localization of spatially distributed brain sources after a tensor-based preprocessing of interictal epileptic EEG data," in *Proc. 37th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Milan, Italy, Aug. 2015, pp. 6995–6998.
[77] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.
[78] L. Grasedyck, "Hierarchical singular value decomposition of tensors," *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 4, pp. 2029–2054, 2010.
[79] L. Karlsson, D. Kressner, and A. Uschmajew, "Parallel algorithms for tensor completion in the CP format," *Parallel Comput.*, vol. 57, pp. 222–234, Sep. 2016.
[80] L. Luo, Y. Xie, Z. Zhang, and W.-J. Li, "Support matrix machines," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 938–947.
[81] L. R. Tucker, "Implications of factor analysis of three-way matrices for measurement of change," in *Problems Measuring Change*, C. W. Harris, Ed. Madison, WI, USA: Univ. Wisconsin Press, 1963, pp. 122–137.
[82] L. Sorber, I. Domanov, M. Van Barel, and L. De Lathauwer, "Exact line and plane search for tensor optimization," *Comput. Optim. Appl.*, vol. 63, no. 1, pp. 121–142, 2016.
[83] L. Yuan, Q. Zhao, and J. Cao, "High-order tensor completion for data recovery via sparse tensor-train optimization," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Calgary, AB, Canada, Apr. 2018, pp. 1258–1262.
[84] L. Zhai, Y. Zhang, H. Lv, S. Fu, and H. Yu, "Multiscale tensor dictionary learning approach for multispectral image denoising," *IEEE Access*, vol. 6, pp. 51898–51910, 2018.
[85] M. Bebendorf, C. Kuske, and R. Venn, "Wideband nested cross approximation for Helmholtz problems," *Numerische Mathematik*, vol. 130, no. 1, pp. 1–34, 2015.
[86] M. Bachmayr, R. Schneider, and A. Uschmajew, "Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations," *Found. Comput. Math.*, vol. 16, no. 6, pp. 1423–1472, 2016.
[87] M. Espig, M. Schuster, A. Killaitis, N. Waldren, P. Whnert, S. Handschuh, and H. Auer, "TensorCalculus library," Tech. Rep., 2012.
[88] M. Ghassemi, Z. Shakeri, A. D. Sarwate, and W. U. Bajwa, "STARK: Structured dictionary learning through rank-one tensor recovery," in *Proc. IEEE 7th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP)*, Curacao, Netherlands Antilles, Dec. 2017, pp. 1–5.
[89] M. Hou, "Tensor-based regression models and applications," Ph.D. dissertation, Laval Univ., Quebec City, QC, Canada, 2017.
[90] M. Hou, Y. Wang, and B. Chaib-draa, "Online local Gaussian process for tensor-variate regression: Application to fast reconstruction of limb movements from brain signal," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Brisbane, QLD, Australia, Apr. 2015, pp. 5490–5494.
[91] M. Hou, Q. Zhao, B. Chaib-Draa, and A. Cichocki, "Common and discriminative subspace kernel-based multiblock tensor partial least squares regression," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1673–1679.
[92] M. Steinlechner, "Riemannian optimization for solving high-dimensional problems with low-rank tensor structure," Ph.D. dissertation, 2016.
[93] M. W. Mahoney, M. Maggioni, and P. Drineas, "Tensor-CUR decompositions for tensor-based data," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 957–987, 2008.
[94] N. Cohen and A. Shashua, "Inductive bias of deep convolutional networks through pooling geometry," 2016, *arXiv:1605.06743*. [Online]. Available: https://arxiv.org/abs/1605.06743
[95] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, "Tensor decomposition for signal processing and machine learning," *IEEE Trans. Signal Process.*, vol. 65, no. 13, pp. 3551–3582, Jal. 2017.
[96] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.
[97] N. H. Nguyen, P. Drineas, and T. D. Tran, "Tensor sparsification via a bound on the spectral norm of random tensors," 2015, *arXiv:1005.4732*. [Online]. Available: https://arxiv.org/abs/1005.4732
[98] N. Kargas and N. D. Sidiropoulos, "Completing a joint PMF from projections: A low-rank coupled tensor factorization approach," in *Proc. Inf. Theory Appl. Workshop (ITA)*, San Diego, CA, USA, Feb. 2017, pp. 1–6.
[99] N. Lee and A. Cichocki, "Fundamental tensor operations for large-scale data analysis using tensor network formats," *Multidimensional Syst. Signal Process.*, vol. 29, no. 3, pp. 921–960, 2018.
[100] N. Schuch, I. Cirac, and D. Pérez-García, "PEPS as ground states: Degeneracy and topology," *Ann. Phys.*, vol. 325, no. 10, pp. 2153–2192, 2010.

[101] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition," *SIAM J. Sci. Comput.*, vol. 34, no. 2, pp. A1027–A1052, 2012.

[102] P. D. Hoff, "Multilinear tensor regression for longitudinal relational data," *Ann. Appl. Statist.*, vol. 9, no. 3, pp. 1169–1193, 2015.

[103] P. G. Constantine, D. F. Gleich, Y. Hou, and J. Templeton, "Model reduction with mapreduce-enabled tall and skinny singular value decomposition," *SIAM J. Sci. Comput.*, vol. 36, no. 5, pp. S166–S191, 2014.

[104] P. M. Kroonenberg, *Applied Multiway Data Analysis*. New York, NY, USA: Wiley, 2008.

[105] Q. Li, G. An, and Q. Ruan, "3D facial expression recognition using orthogonal tensor marginal Fisher analysis on geometric maps," in *Proc. Int. Conf. Wavelet Anal. Pattern Recognit. (ICWAPR)*, Ningbo, China, Jul. 2017, pp. 65–71.

[106] Q. Li and G. Tang, "Convex and nonconvex geometries of symmetric tensor factorization," in *Proc. 51st Asilomar Conf. Signals, Syst., Comput.*, Pacific Grove, CA, USA, Oct./Nov. 2017, pp. 305–309.

[107] Q. Shi, Y.-M. Cheung, Q. Zhao, and H. Lu, "Feature extraction for incomplete data via low-rank tensor decomposition with feature regularization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 6, pp. 1803–1817, Jun. 2019.

[108] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "A tensor-train deep computation model for industry informatics big data feature learning," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3197–3204, Jul. 2018.

[109] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor ring decomposition," 2016, *arXiv:1606.05535*. [Online]. Available: https://arxiv.org/abs/1606.05535

[110] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an 'explanatory' multimodal factor analysis," UCLA Working Papers in Phonetics, Tech. Rep., 1970, pp. 1–84, vol. 16.

[111] R. Kountchev and R. Kountcheva, "Truncated hierarchical SVD for image sequences, represented as third order tensor," in *Proc. 8th Int. Conf. Inf. Technol. (ICIT)*, Amman, Jordan, May 2017, pp. 166–173.

[112] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Ann. Phys.*, vol. 349, pp. 117–158, Oct. 2014.

[113] R. Yu and Y. Liu, "Learning from multiway data: Simple and efficient tensor regression," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, 2016, pp. 373–381.

[114] R. Zhao and Q. Wang, "Learning separable dictionaries for sparse tensor representation: An online approach," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 3, pp. 502–506, Mar. 2019.

[115] R. Zdunek and K. Fonal, "Randomized nonnegative tensor factorization for feature extraction from high-dimensional signals," in *Proc. 25th Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Maribor, Slovenia, Jun. 2018, pp. 1–5.

[116] S. A. Vorobyov, Y. Rong, N. D. Sidiropoulos, and A. B. Gershman, "Robust iterative fitting of multilinear models," *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2678–2689, Aug. 2005.

[117] S. Chen and S. A. Billings, "Representations of non-linear systems: The NARMAX model," *Int. J. Control*, vol. 49, no. 3, pp. 1013–1032, 1989.

[118] S. E. Sofuoglu and S. Aviyente, "A two-stage approach to robust tensor decomposition," in *Proc. IEEE Stat. Signal Process. Workshop (SSP)*, Freiburg, Germany, Jun. 2018, pp. 831–835.

[119] S. Han and P. Woodford, "Comparison of dimension reduction methods using polarimetric SAR images for tensor-based feature extraction," in *Proc. 12th Eur. Conf. Synth. Aperture Radar (EUSAR)*, Aachen, Germany, Jun. 2018, pp. 1–6.

[120] S. Kallam, S. M. Basha, D. S. Rajput, R. Patan, B. Balamurugan, and S. A. K. Basha, "Evaluating the performance of deep learning techniques on classification using tensor flow application," in *Proc. Int. Conf. Adv. Comput. Commun. Eng. (ICACCE)*, Paris, France, Jun. 2018, pp. 331–335.

[121] S. K. Biswas and P. Milanfar, "Linear support tensor machine with LSK channels: Pedestrian detection in thermal infrared images," *IEEE Trans. Image Process.*, vol. 26, no. 9, pp. 4229–4242, Sep. 2017.

[122] S. Savvaki, G. Tsagkatakis, A. Panousopoulou, and P. Tsakalides, "Matrix and tensor completion on a human activity recognition framework," *IEEE J. Biomed. Health Inform.*, vol. 21, no. 6, pp. 1554–1561, Nov. 2017.

[123] S. V. Dolgov and D. V. Savostyanov, "Alternating minimal energy methods for linear systems in higher dimensions," *SIAM J. Sci. Comput.*, vol. 36, no. 5, pp. A2248–A2271, 2014.

[124] S. Yang, M. Wang, Z. Feng, Z. Liu, and R. Li, "Deep sparse tensor filtering network for synthetic aperture radar images classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 8, pp. 3919–3924, Aug. 2018.

[125] T. D. Pham and H. Yan, "Tensor decomposition of gait dynamics in Parkinson's disease," *IEEE Trans. Biomed. Eng.*, vol. 65, no. 8, pp. 1820–1827, Aug. 2018.

[126] T. D. Nguyen, T. Tran, D. Phung, and S. Venkatesh, "Tensor-variate restricted Boltzmann machines," in *Proc. AAAI*, 2015, pp. 2887–2893.

[127] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Rev.*, vol. 51, no. 3, pp. 455–500, 2009.

[128] T.-X. Jiang, T.-Z. Huang, X.-L. Zhao, L.-J. Deng, and Y. Wang, "A novel tensor-based video rain streaks removal approach via utilizing discriminatively intrinsic priors," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 2818–2827.

[129] T.-L. Chen, D. D. Chang, S.-Y. Huang, H. Chen, C. Lin, and W. Wang, "Integrating multiple random sketches for singular value decomposition," 2016, *arXiv:1608.08285*. [Online]. Available: https://arxiv.org/abs/1608.08285

[130] T. Wu, A. R. Benson, and D. F. Gleich, "General tensor spectral co-clustering for higher-order data," 2016, *arXiv:1603.00395*. [Online]. Available: https://arxiv.org/abs/1603.00395

[131] T. Yokota, N. Lee, and A. Cichocki, "Robust multilinear tensor rank estimation using higher order singular value decomposition and information criteria," *IEEE Trans. Signal Process.*, vol. 65, no. 5, pp. 1196–1206, Mar. 2017.

[132] V. A. Kazeev, B. N. Khoromskij, and E. E. Tyrtyshnikov, "Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity," *SIAM J. Sci. Comput.*, vol. 35, no. 3, pp. A1511–A1536, 2013.

[133] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, 2009, Art. no. 15.

[134] V. de Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1084–1127, 2008.

[135] V. Giovannetti, S. Montangero, and R. Fazio, "Quantum multiscale entanglement renormalization ansatz channels," *Phys. Rev. Lett.*, vol. 101, no. 18, 2008, Art. no. 180503.

[136] V. Kuleshov, A. Chaganty, and P. Liang, "Tensor factorization via matrix factorization," in *Proc. 18th Int. Conf. Artif. Intell. Statist.*, 2015, pp. 507–516.

[137] V. Tresp, C. Esteban, Y. Yang, S. Baier, and D. Krompaß, "Learning with memory embeddings," 2015, *arXiv:1511.07972*. [Online]. Available: https://arxiv.org/abs/1511.07972

[138] W. Austin, G. Ballard, and T. G. Kolda, "Parallel tensor compression for large-scale scientific data," 2015, *arXiv:1510.06689*. [Online]. Available: https://arxiv.org/abs/1510.06689

[139] W. Chu and Z. Ghahramani, "Probabilistic models for incomplete multi-dimensional arrays," in *Proc. 12th Int. Conf. Artif. Intell. Statist.*, vol. 5, 2009, pp. 89–96.

[140] W. de Launey and J. Seberry, "The strong Kronecker product," *J. Combinat. Theory, Ser. A*, vol. 66, no. 2, pp. 192–213, 1994.

[141] W. Guo, I. Kotsia, and I. Patras, "Tensor learning for regression," *IEEE Trans. Image Process.*, vol. 21, no. 2, pp. 816–827, Feb. 2012.

[142] W. Hackbusch and S. Kühn, "A new scheme for the tensor representation," *J. Fourier Anal. Appl.*, vol. 15, no. 5, pp. 706–722, Oct. 2009.

[143] W. Hu, J. Gao, J. Xing, C. Zhang, and S. Maybank, "Semi-supervised tensor-based graph embedding learning and its application to visual discriminant tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 1, pp. 172–188, Jan. 2017.

[144] X. Zhao, H. Shi, M. Lv, and L. Jing, "Least squares twin support tensor machine for classification," *J. Inf. Comput. Sci.*, vol. 11, no. 12, pp. 4175–4189, 2014.

[145] X. Deng, P. Jiang, X. Peng, and C. Mi, "An intelligent outlier detection method with one class support Tucker machine and genetic algorithm toward big sensor data in Internet of Things," *IEEE Trans. Ind. Electron.*, vol. 66, no. 6, pp. 4672–4683, Jun. 2019.

[146] X. He, D. Cai, and P. Niyogi, "Tensor subspace analysis," in *Proc. Annu. Conf. Neural Inf. Process. Syst*, 2006, pp. 499–506.

[147] X. Xu, N. Zhang, Y. Yan, and Q. Shen, "Application of support higher-order tensor machine in fault diagnosis of electric vehicle range-extender," in *Proc. Chin. Autom. Congr. (CAC)*, Jinan, China, Oct. 2017, pp. 6033–6037.

[148] X. Xu, Q. Wu, S. Wang, J. Liu, J. Sun, and A. Cichocki, "Whole brain fMRI pattern analysis based on tensor neural network," *IEEE Access*, vol. 6, pp. 29297–29305, 2018.
[149] X. Zhang, "A nonconvex relaxation approach to low-rank tensor completion," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 6, pp. 1659–1671, Jun. 2019.
[150] Y. Du, G. Han, Y. Quan, Z. Yu, H.-S. Wong, C. L. P. Chen, and J. Zhang, "Exploiting global low-rank structure and local sparsity nature for tensor completion," *IEEE Trans. Cybern.*, vol. 49, no. 11, pp. 3898–3910, Nov. 2019.
[151] Y. Huang, W. Wang, L. Wang, and T. Tan, "Conditional high-order Boltzmann machines for supervised relation learning," *IEEE Trans. Image Process.*, vol. 26, no. 9, pp. 4297–4310, Sep. 2017.
[152] Y.-J. Kao, Y.-D. Hsieh, and P. Chen, "Uni10: An open-source library for tensor network algorithms," *J. Phys., Conf. Ser.*, vol. 640, no. 1, 2015, Art. no. 012040.
[153] Y. Liu, "Low-rank tensor regression: Scalability and applications," in *Proc. IEEE 7th Int. Workshop Comput. Adv. Multi-Sensor Adapt. Process. (CAMSAP)*, Curacao, Netherlands Antilles, Dec. 2017, pp. 1–5.
[154] Y. Wang, H.-Y. Tung, A. Smola, and A. Anandkumar, "Fast and guaranteed tensor decomposition via sketching," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 991–999.
[155] Y. Wang, W. Zhang, Z. Yu, Z. Gu, H. Liu, Z. Cai, C. Wang, and S. Gao, "Support vector machine based on low-rank tensor train decomposition for big data applications," in *Proc. 12th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Siem Reap, Cambodia, Jun. 2017, pp. 850–853.
[156] Y. W. Chen, K. Guo, and Y. Pan, "Robust supervised learning based on tensor network method," in *Proc. 33rd Youth Acad. Annu. Conf. Chin. Assoc. Automat. (YAC)*, Nanjing, China, May 2018, pp. 311–315.
[157] Y. Xiang, Q. Jiang, J. He, X. Jin, L. Wu, and S. Yao, "The advance of support tensor machine," in *Proc. IEEE 16th Int. Conf. Softw. Eng. Res., Manage. Appl. (SERA)*, Kunming, China, Jun. 2018, pp. 121–128.
[158] Y. Zhang and R. Barzilay, "Hierarchical low-rank tensors for multilingual transfer parsing," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1857–1867.
[159] Z. Chen, K. Batselier, J. A. K. Suykens, and N. Wong, "Parallelized tensor train learning of polynomial classifiers," 2016, *arXiv:1612.06505*. [Online]. Available: https://arxiv.org/abs/1612.06505
[160] Z. Chen, B. Yang, and B. Wang, "Hyperspectral target detection: A preprocessing method based on tensor principal component analysis," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Valencia, Spain, 2018, pp. 2753–2756.
[161] Z. Chen, K. Batselier, J. A. K. Suykens, and N. Wong, "Parallelized tensor train learning of polynomial classifiers," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4621–4632, Oct. 2018.
[162] Z.-C. Gu, M. Levin, B. Swingle, and X.-G. Wen, "Tensor-product representations for string-net condensed states," *Phys. Rev. B, Condens. Matter*, vol. 79, no. 8, 2009, Art. no. 085118.
[163] Z. Fang, X. Yang, L. Han, and X. Liu, "A sequentially truncated higher order singular value decomposition-based algorithm for tensor completion," *IEEE Trans. Cybern.*, vol. 49, no. 5, pp. 1956–1967, May 2019.
[164] Z. Hao, L. He, B. Chen, and X. Yang, "A linear support higher-order tensor machine for classification," *IEEE Trans. Image Process.*, vol. 22, no. 7, pp. 2911–2920, Jul. 2013.
[165] Z. Zhang and S. Aeron, "Exact tensor completion using t-SVD," *IEEE Trans. Signal Process.*, vol. 65, no. 6, pp. 1511–1526, Mar. 2015.

YUWANG JI is currently pursuing the master's degree with the National Engineering Laboratory for Mobile Network Security, Wireless Technology Innovation Institute, Beijing University of Posts and Telecommunications (BUPT). His current research interests include the tensor application in machine learning and time series analysis.

QIANG WANG received the Ph.D. degree in communication engineering from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2008. Since 2008, he has been with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, where he is currently an Associate Professor. He participated in many national projects such as NSFC, 863, and so on. His research interests include information theory, machine learning, wireless communications, VLSI, and statistical inference.

XUAN LI is currently pursuing the master's degree with the National Engineering Laboratory for Mobile Network Security, Wireless Technology Innovation Institute, Beijing University of Posts and Telecommunications (BUPT). Her current research interests include UAV-assisted networks and reinforcement learning.

JIE LIU is currently pursuing the master's degree with the National Engineering Laboratory for Mobile Network Security, Wireless Technology Innovation Institute, Beijing University of Posts and Telecommunications (BUPT). Her current research interests include time series prediction and reinforcement learning.