Quantized Tensor Neural Network

YUAN GAO, Huazhong University of Science and Technology

LAURENCE T. YANG, Huazhong University of Science and Technology and

St. Francis Xavier University

DEHUA ZHENG and JING YANG, Huazhong University of Science and Technology

YALIANG ZHAO, Henan University and St. Francis Xavier University

Tensor network as an effective computing framework for efficient processing and analysis of high-dimensional data has been successfully applied in many fields. However, the performance of traditional tensor networks still cannot match the strong fitting ability of neural networks, so some data processing algorithms based on tensor networks cannot achieve the same excellent performance as deep learning models. To further improve the learning ability of tensor network, we propose a quantized tensor neural network in this article (QTNN), which integrates the advantages of neural networks and tensor networks, namely, the powerful learning ability of neural networks and the simplicity of tensor networks. The QTNN model can be further regarded as a generalized multilayer nonlinear tensor network, which can efficiently extract low-dimensional features of the data while maintaining the original structure information. In addition, to more effectively represent the local information of data, we introduce multiple convolution layers in QTNN to extract the local features. We also develop a high-order back-propagation algorithm for training the parameters of QTNN. We conducted classification experiments on multiple representative datasets to further evaluate the performance of proposed models, and the experimental results show that QTNN is simpler and more efficient while compared to the classic deep learning models.

CCS Concepts: • **Computing methodologies** → *Machine learning; Machine learning approaches; Neural networks*;

Additional Key Words and Phrases: Big data, deep learning, tensor, tensor neural networks, high-order error back-propagation algorithm, data classification

ACM Reference format:

Yuan Gao, Laurence T. Yang, Dehua Zheng, Jing Yang, and Yaliang Zhao. 2022. Quantized Tensor Neural Network. *ACM/IMS Trans. Data Sci.* 2, 4, Article 33 (February 2022), 18 pages.

<https://doi.org/10.1145/3491255>

This work was supported in part by the National Natural Science Foundation of China (No. 61802112), the Henan Key Research and Development and Promotion Project (No. 212102310302), and the State Scholarship Fund of China.

Authors' addresses: Y. Gao, D. Zheng, and J. Yang, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China; emails: yuang@hust.edu.cn, dwardz@163.com, jing_yang@hust.edu.cn; L. T. Yang, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China, and Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G2W5, Canada; email: lyyang@gmail.com; Y. Zhao (corresponding author), Henan Key Laboratory of Big Data Analysis and Processing, School of Computer and Information Engineering, Henan University, Kaifeng 475004, China, and with Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G2W5, Canada; email: ylzhao@henu.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.

2577-3224/2022/02-ART33 \$15.00

<https://doi.org/10.1145/3491255>

1 INTRODUCTION

In recent years, the rapid development of information technology, especially the development of technologies represented by Cloud computing, Big data, Internet of Things, Artificial intelligence, and so on, has made smart devices such as mobile phones, PCs, and smart sensors visible everywhere, and the source and quantity of data are growing at an unprecedented rate. In addition to the typical 4V characteristics of big data (Volume, Velocity, Variety, Value), diversified data sources have created the characteristics of big data such as heterogeneity, high-dimension, and the internal structure and relationship of these data are also more complicated [21]. In addition, the data in modern applications has gradually presented multi-dimensional characteristics, which to some extent makes it difficult for traditional vector or matrix-based machine learning algorithms to deal with multi-dimensional data.

Tensor as a natural multidimensional generalization of vector and matrix is an effective tool for the representation and analysis of massive multi-dimensional data [12]. In recent years, tensor models have been successfully applied to many fields, such as chemistry, neuroscience, computer vision, text mining, clustering, social networks [10, 25], and so on. However, tensor volume can easily become prohibitively big as the number of dimensions increase, thus requiring enormous computational and memory resources to process such data. Feature extraction and dimension reduction of high-dimensional data as an important task in the field of data mining has led many scholars to propose related methods for high-dimensional data based on low-rank tensor networks, which allow for huge data tensors to be approximated (compressed) by compact low-rank core tensors, effectively alleviating the curse of dimensionality of high-dimensional data [7, 16]. In addition, some multi-linear analysis methods for image classification based on tensor models are proposed in Reference [3], which models face images involving multiple factors into high-order tensors to extract compact low-dimensional features. Non-negative tensor decomposition models are also proposed to mine potential patterns and components that have stronger interpretability [18], such as image classification [17] and social network mining [6]. Moreover, some manifold algorithms for dimensional reduction of high-dimensional dataset based on tensor models have also been developed in Reference [14]. Through establishing potential correlation patterns among the heterogeneous data from multiple sources, coupled matrix and tensor decomposition further realizes joint analysis of data and effectively enhances knowledge discovery [1]. More generally, the above-mentioned methods rely on the inherent internal structure information of the original data and the global characteristics of the entire dataset (sample similarity) to a large extent and ignore the local structure information of the high-dimensional data, so it is difficult to improve their learning ability.

Deep neural networks are widely used in various fields with their powerful learning and representation capabilities, and they can effectively capture the high-level abstract features behind the data through a deep network architecture. The most representative ones are convolutional neural networks, including GoogleNet [20], VGG [19], ResNet [8] and so on, which have achieved great success in the field of computer vision. However, as the number of network layers continues to increase, the parameter scale of these networks is getting larger, making it difficult to deploy models on small embedded devices [9, 24]. In recent years, some tensor models have been successfully integrated into deep neural networks to improve the structure of the network. The convolutional neural network was accelerated by using fine-tuned CP decomposition [13]. Tensor regression layers are also introduced to CNN to achieve reduction of network parameters while maintaining the multi-linear structure of the hidden layer features [11]. To further reduce the storage requirements of neural network parameters, tensor train model is applied to the network by representing the parameters of the full connection layer in TT form [15, 22]. Furthermore, some research using tensors to explore the interpretability of neural network models is also proposed in References [2, 5].

However, most of these existing methods use the low-rank structure of the tensor network to reduce the parameters of the neural network, instead of really improving the learning ability of the tensor network. To extend the neural network to the high-dimensional tensor space, **tensor factorized neural network (TFNN)** directly builds a multi-layer nonlinear tensor network by stacking multiple Tucker regression layers [4]. In addition, an unsupervised deep computation model based on tensor is proposed to learn features of heterogeneous big data in Reference [23].

In this article, we further propose a **quantized tensor neural network (QTNN)** to improve the learning ability of the tensor network, which effectively integrates the high-order convolution operations into the tensor network to learn the local features of high-dimensional data. First, we generalize the standard convolution operation to more general high-order scenarios and propose a high-order convolution operation to capture the local information of high-dimensional data. Then, the high-order tensors containing local information are further input into a multi-layer nonlinear tensor network for feature learning and classification. Moreover, we also develop a high-order error back-propagation algorithm based on tensor networks for parameter optimization. Finally, we conducted classification experiments on three real-world datasets to verify the performance of the proposed models, and the experimental results fully demonstrate that the QTNN model integrating neural networks and tensor networks has superior performance. The contributions of this article can be summarized as follows:

- (1) Considering that the traditional tensor networks have limited learning ability, and it is difficult for them to learn the hidden features of data sufficiently, we propose a multi-layer nonlinear tensor network, which further combines the idea of neural network to learn the factor matrices of each layer by optimizing the objective function while maintaining the original structure of data. We also theoretically demonstrate that it is equivalent to a traditional NN model, but more efficient.
- (2) To further improve the classification performance of the tensor network, we integrate high-order convolution operations into the nonlinear tensor network to learn the local neighborhood features of the original data and propose a **quantized tensor neural network (QTNN)**. Compared with the classic CNN models, QTNN can learn the hidden features of data more efficiently and has fewer parameters.
- (3) To further optimize the parameters in the network, we generalize the classical back-propagation algorithm to higher-order scenarios and proposed a high-order error back-propagation algorithm based on tensor neural network, which also realizes the transformation of the traditional optimization algorithms from vector space to higher-order tensor space.

We organize the rest of this article as follows: In Section 2, we review some related preliminaries and notations. Some basic definitions are provided in Section 3, which are necessary for further constructing the tensor neural networks. Section 4 presents two tensor neural network models and a corresponding high-order error back-propagation algorithm. The complexity of the proposed models is analyzed in Section 5. After that, the experiments conducted on three representative datasets are provided in Section 6. Finally, we give the conclusion of this article in Section 7.

2 PRELIMINARIES

2.1 Tensor and Tensor Decomposition

Tensors, as natural multi-dimensional generalizations of vectors and matrices, have been successfully applied to several domains such as computer vision, neuroscience, and pattern recognition.

In this section, we briefly review some basic knowledge of tensors, which is also a prerequisite for understanding tensor neural networks.

Tucker decomposition (also called higher-order SVD) provides a more general factorized format for high-dimensional tensors, which can be regarded as the higher-order extension of the singular value decomposition, and the Tucker model can be formulated as follows:

$$\begin{aligned}\mathcal{X} &\cong \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} \mathcal{G}_{r_1, \dots, r_N} a_{r_1}^{(1)} \circ a_{r_2}^{(2)} \cdots \circ a_{r_N}^{(N)} \\ &= \mathcal{G} \times_1 A^{(1)} \times_2 A^{(2)} \cdots \times_N A^{(N)},\end{aligned}\quad (1)$$

where $\mathcal{X} \in R^{I_1 \times I_2 \cdots \times I_N}$ is the original data tensor, $\mathcal{G} \in R^{R_1 \times R_2 \cdots \times R_N}$ is core tensor, and $A^{(n)} = [a_1^{(n)}, a_2^{(n)}, \dots, a_{R_n}^{(n)}] \in R^{I_n \times R_n}$, $R_n \ll I_n$, is mode- n factor matrix, which provides the principal component of mode n , $n = 1, 2, \dots, N$, and the potential complex interaction pattern between principal components of different modes modeled by the core tensor.

Actually, the above Tucker decomposition result could be further transformed into the following form to establish the connection between the input data and the compressed low-dimensional core tensor:

$$\mathcal{G} = \mathcal{X} \times_1 A^{(1)\dagger} \times_2 A^{(2)\dagger} \cdots \times_N A^{(N)\dagger},$$

where $A^{(n)\dagger} = (A^{(n)T} A^{(n)})^{-1} A^{(n)T}$, $1 \leq n \leq N$, are the pseudo-inverses of factor matrices. Without loss of generality, $\mathcal{G}$ is compressed feature tensor with low-dimension of original data tensor $\mathcal{X}$.

Alternatively, the modular multiplication transformation from $\mathcal{X}$ to $\mathcal{G}$ can be interpreted as a convolution operation. Each element of $\mathcal{G}$ is obtained by inner product of the original data $\mathcal{X}$ and the rank-1 convolution kernel tensor. The total number of convolution kernels is $R_1 R_2 \dots R_N$, and they are all of the same order and dimension as the original data. Different from the convolution operation in classical CNN model that has a local receptive field, high-order Tucker model has a global receptive field. For example,

$$\mathcal{G} = \mathcal{X} \times_1 A^{(1)} \times_2 A^{(2)} \cdots \times_N A^{(N)}.$$

consider each element of $\mathcal{G}$, then we have following expression:

$$\begin{aligned}\mathcal{G}_{r_1, r_2, \dots, r_N} &= \sum_{i_1, i_2, \dots, i_N} \mathcal{X}_{i_1, i_2, \dots, i_N} A_{r_1, i_1}^{(1)} A_{r_2, i_2}^{(2)} \cdots A_{r_N, i_N}^{(N)} \\ &= \sum_{i_1, i_2, \dots, i_N} \mathcal{X}_{i_1, i_2, \dots, i_N} \mathcal{A}_{i_1, i_2, \dots, i_N} \\ &= \langle \mathcal{X}, \mathcal{A}_{r_1, r_2, \dots, r_N} \rangle,\end{aligned}\quad (2)$$

where $\mathcal{A}_{r_1, r_2, \dots, r_N} = A_{r_1, :}^{(1)} \circ A_{r_2, :}^{(2)} \cdots \circ A_{r_N, :}^{(N)} \in R^{I_1 \times I_2 \cdots \times I_N}$ is an N -order kernel tensor indirectly generated by N factor matrices.

3 DEFINITIONS AND NOTATIONS

To effectively represent the local structure information of data, we define the following high-order quantization and convolution operations to extract the local neighborhood features of high-order tensors. We also introduce a nonlinear Tucker layer to extract the hidden features while maintaining the original structure of data. In fact, the operations defined below can be regarded as the high-order generalizations of operations defined in the standard CNN, and some examples are provided in Figure 1.

Fig. 1. Graphic representation of some defined operations.

Definition 3.1 (High-order Quantification). Given an N -order tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$, and $I_n = I'_n \times J_n, 1 \leq n \leq N$, then $\mathcal{X}$ can be represented as:

$$\mathcal{X}' \in R^{I'_1 \times I'_2 \times \dots \times I'_N \times J_1 \times J_2 \times \dots \times J_N},$$

where the first N orders of $\mathcal{X}'$ are used to represent the local information of the original data, and the remaining N orders are used to indicate the number of data blocks divided in each dimension of the original data, and the original data is totally divided into $J_1 J_2 \dots J_N$ blocks with the size of $I'_1 \times I'_2 \times \dots \times I'_N$.

Definition 3.2 (Generalized High-order Quantification). Given an N -order tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$, and fix the size of each data block as $I'_1 \times I'_2 \times \dots \times I'_N$, the strides are $\{s_1, s_2, \dots, s_N\}$, then there are total $J'_1 J'_2 \dots J'_N$ intersecting blocks, $J'_n = \lceil \frac{I_n - I'_n}{s_n} \rceil + 1, 1 \leq n \leq N$, and $\mathcal{X}$ can be further represented as:

$$\mathcal{X}' \in R^{I'_1 \times I'_2 \times \dots \times I'_N \times J'_1 \times J'_2 \times \dots \times J'_N},$$

where the first N orders represent size of each block, and the last N orders represent the number of data blocks divided in each dimension of the original data.

Definition 3.3 (High-order Convolution Operation). Given an N -order tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$ and an N -order convolution kernel tensor $\mathcal{W}$ with the size of $k_1 \times k_2 \times \dots \times k_n$, the strides and padding size are $\{s_1, s_2, \dots, s_N\}, \{p_1, p_2, \dots, p_N\}$, respectively, then the convolution of two tensors is defined as follows:

$$\mathcal{F} = \mathcal{W} * \mathcal{X},$$

where the feature tensor $\mathcal{F} \in R^{J_1 \times J_2 \times \dots \times J_N}$, $J_n = \lceil \frac{I_n + 2p_n - k_n}{s_n} \rceil + 1, 1 \leq n \leq N$, and each element of feature tensor contains $k_1 \times k_2 \times \dots \times k_n$ spatial neighborhood information of original tensor.

Definition 3.4 (High-order Pooling Operation). Given an N -order feature tensor $\mathcal{F} \in R^{J_1 \times J_2 \times \dots \times J_N}$, and the pooling stride along each order is $s = \{s_1, s_2, \dots, s_N\}$, then the pooling operation of $\mathcal{F}$ is defined as follows:

$$\mathcal{F}' = \text{Pooling}(\mathcal{F}, s),$$

where the feature tensor $\mathcal{F}' \in R^{K_1 \times K_2 \times \dots \times K_N}$, $K_n = J_n / s_n, 1 \leq n \leq N$.

Definition 3.5 (Nonlinear Tucker Layer). Given an N -order input tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$, the nonlinear Tucker layer is defined as follows:

$$\mathcal{Z} = h\{\mathcal{X} \times_1 U^{(1)} \times_2 U^{(2)} \dots \times_N U^{(N)} + \mathcal{E}\}, \quad (3)$$

where $\mathcal{Z} \in R^{J_1 \times J_2 \times \dots \times J_N}$ is potential low-dimensional feature of $\mathcal{X}$, $U^{(n)} \in R^{J_n \times I_n}$, $J_n \ll I_n$, $1 \leq n \leq N$, are parameter matrices, $\mathcal{E} \in R^{J_1 \times J_2 \times \dots \times J_N}$ is a high-order bias tensor, and $h : R^{J_1 \times J_2 \times \dots \times J_N} \rightarrow R^{J_1 \times J_2 \times \dots \times J_N}$ is a nonlinear activation function.

Considering that the size of the parameter tensor $\mathcal{E}$ increases exponentially with the increase of the orders, we can further represent the bias tensor in the format of CP to reduce the amount of parameters, that is, $\mathcal{E} = [B^{(1)}, \dots, B^{(N)}] = \sum_{k=1}^K b_k^{(1)} \circ b_k^{(2)} \dots \circ b_k^{(N)}$, and $B^{(n)} = [b_1^{(n)}, \dots, b_K^{(n)}] \in R^{J_n \times K}$, $1 \leq n \leq N$. Actually, the nonlinear Tucker layer is equivalent to the full connection layer of DNN, and the parameter matrix W of DNN can be generated from N factor matrices in the following way:

$$vec(\mathcal{Z}) = h\{(U^{(N)} \otimes \dots \otimes U^{(1)})vec(\mathcal{X}) + vec(\mathcal{E})\}, \quad (4)$$

where $\otimes$ is the Kronecker product, $W = U^{(N)} \otimes U^{(N-1)} \dots \otimes U^{(1)} \in R^{J_1 J_2 \dots J_N \times I_1 I_2 \dots I_N}$, and $b = vec(\mathcal{E}) \in R^{J_1 J_2 \dots J_N}$. However, it is obvious that the weights generated by the N factor matrices have fewer parameters than the weights directly defined by DNN, that is, the number of parameters goes from $J_1 \dots J_N I_1 \dots I_N$ to $\sum_{n=1}^N J_n \times I_n$. In addition, nonlinear Tucker layer can mine the hidden features of the data more efficiently while maintaining the original structural information of the data.

Definition 3.6 (Tensor Contraction Layer). Given an N -order tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$, and an $N+1$ -order parameter tensor $\mathcal{W} \in R^{I_1 \times I_2 \times \dots \times I_N \times C}$, then the tensor contraction operation between $\mathcal{X}$ and $\mathcal{W}$ is defined as follows:

$$\langle \mathcal{X}, \mathcal{W} \rangle_c = \mathcal{X} \times_{i_1, \dots, i_N} \mathcal{W} + \mathbf{b} = \sum_{i_1, i_2, \dots, i_N} \mathcal{X}_{i_1, i_2, \dots, i_N} \mathcal{W}_{i_1, i_2, \dots, i_N, \cdot} + \mathbf{b}, \quad (5)$$

where $\mathbf{b} \in R^C$ is a bias vector.

4 TENSOR NEURAL NETWORK CONSTRUCTING

4.1 Tensor Neural Network

The main ideal of TNN is to extend the neural network models to a more general high-dimensional situation through the tensor decomposition model and establish a close correlation between the tensor network and the neural network [4]. The general TNNs of 1-order, 2-order, and N -order are provided in Figure 2, and it is obvious that the 1-order TNN is equivalent to the NN model. Combining Equation (4), 2-order, N -order, or higher-order TNN models could be converted into standard NN models through vectorization processing. Actually, each layer of TNN is composed of the nonlinear Tucker layer.

Considering an N order input tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$, $\mathcal{A}^{(1)} \in R^{J_1 \times J_2 \times \dots \times J_N}$ is corresponding core tensor, and $\mathcal{Z}^{(1)}$ is the activated feature tensor in first layer. In this way, each layer of the TNN is essentially a process of factor analysis or structural analysis of the input data. For the first case, the N -order core tensor $\mathcal{A}^{(1)}$ is obtained by N linear transformations of the original data, and the factor matrix $U^{(n)}$ can be regarded as the main factor components of mode- n , $1 \leq n \leq N$, which are used to project $\mathcal{X}$ into a core tensor $\mathcal{A}^{(1)}$, then get $\mathcal{Z}^{(1)}$ through the activation function $h(\cdot)$. For the second case, the nonlinear Tucker layer can be further regarded as a structural analysis

Fig. 2. High-order generalization of the traditional NN models.

Fig. 3. The structure of multi-layer TNN.

process of the original data, and the basic structure tensors $\mathcal{U}^{(k)} \in R^{I_1 \times I_2 \times \dots \times I_N}$, $1 \leq k \leq J_1 J_2 \dots J_N$ are generated by the factor matrices in the following way:

$$\mathcal{U}^{(j)} = U_{j_1, :}^{(1)} \circ U_{j_2, :}^{(2)} \dots \circ U_{j_N, :}^{(N)}, \quad (6)$$

where $j = j_1 + \sum_{n=2}^N (j_n - 1) \prod_{t=1}^{n-1} J_t$, and each element in $\mathcal{A}^{(1)}$ is formulated by inner product, which project X into high-dimensional tensor space through structure tensors. For example, $\mathcal{A}_{j_1, \dots, j_N}^{(1)} = \langle X, \mathcal{U}^{(j)} \rangle$, which means the component of the original data on the structure tensor $\mathcal{U}^{(j)}$. In addition, this high-dimensional tensor space is formulated by $J_1 J_2 \dots J_N$ structure tensors.

As described above, the output of the first hidden layer of the N -order TNN is constructed by N factor matrices $\{U^{(1)}, U^{(2)}, \dots, U^{(N)}\}$, a bias tensor $\mathcal{E}$ and activation functions $h(\cdot)$.

$$\mathcal{Z}^{(1)} = h\{X \times_1 U^{(1)} \times_2 U^{(2)} \dots \times_N U^{(N)} + \mathcal{E}\} \quad (7)$$

As shown in Figure 3, we can further construct a deeper TNN by learning multiple hidden features $\{\mathcal{Z}^{(1)}, \mathcal{Z}^{(2)}, \dots\}$, and $U_n^{(l)}$, $1 \leq n \leq N$ denote N factor matrices of the l th hidden layer, $\mathcal{E}^{(l)}$ is corresponding bias tensor. For each layer of the deep TNN, like a single-layer TNN, it is a process of factor analysis or structural analysis of the high-order data, and N factor matrices in fixed layer are shared for different input data, which is also very reasonable.

4.2 Quantized Tensor Neural Network

For the 2-order TNN, the extracted feature $A^{(1)}$ in the first layer is obtained by bilinear transformation $U^{(1)}$ and $U^{(2)}$, which is significantly different from the typical CNN models. Here, we further propose a quantized tensor neural network, which can effectively use the local structure information of the data, and the classic pooling operation can be indirectly completed by modular multiplication operation on the specified orders. More specifically, the QTNN model of high-dimensional data is mainly divided into three steps. The first step is to extract the high-order neighborhood features of the data through tensor convolution operation. For example, to extract $k_1 \times k_2 \dots \times k_N$ neighborhood features of N -order data tensor $\mathcal{X} \in R^{I_1 \times I_2 \dots \times I_N}$, we use $k_1 k_2 \dots k_N$ convolution kernels to convolve the original data to obtain an $N + 1$ -order tensor $\mathcal{X}' \in R^{I_1 \times I_2 \dots \times I_N \times k_1 \dots k_N}$ and represent it as a $2N$ -order tensor $\mathcal{X}' \in R^{I_1 \times \dots \times I_N \times k_1 \dots k_N}$, then $\mathcal{X}'_{i_1, i_2, \dots, i_N, :, \dots, :} \in R^{k_1 \times \dots \times k_N}$ can be further regarded as the local feature of element $\mathcal{X}_{i_1, i_2, \dots, i_N}$. In fact, the main idea of this operation is to reduce the redundant features in the network by giving the convolution features the same spatial structure information as the original data. The second step is the feature extraction of these small data blocks based on tensor network, and the last step is the feature interaction between the feature blocks.

Given an N -order input $\mathcal{X} \in R^{I_1 \times I_2 \dots \times I_N}$ and its corresponding convolution feature tensor $\mathcal{X}' \in R^{I_1 \times I_2 \dots \times I_N \times k_1 \dots k_N}$, which can also be obtained by multiple convolution layers to expand the field of view of each element. Then the output of the first hidden layer $\mathcal{Z}^{(1)}$ of the network can be obtained by the following formula:

$$\mathcal{Z}^{(1)} = h\{\mathcal{X}' \times_1 U_1^{(1)} \times \dots \times_N U_N^{(1)} \times_{N+1} V_{N+1}^{(1)} \times \dots \times_{2N} V_{2N}^{(1)} \oplus \mathcal{E}^{(1)}\}, \quad (8)$$

where $U_n^{(1)} \in R^{J_n \times I_n}$, $J_n \ll I_n$, $V_{N+n}^{(1)} \in R^{s_n \times k_n}$, $s_n \leq k_n$, $1 \leq n \leq N$, and $\mathcal{E}^{(1)} \in R^{I_1 \times J_2 \times \dots \times J_N}$. Without loss of generality, $U_*^{(1)}$ is the factor matrix that is used to extract the features of data blocks, and $V_*^{(1)}$ is used to perform feature interaction between the data blocks. Equation (8) can be further described as following formats, which is more meaningful for QTNN model.

Actually, the $(k_1 + \sum_{n=2}^N (k_n - 1) \prod_{t=1}^{n-1} k_t)$ -th block of $\mathcal{X}'$ is denoted as $\mathcal{X}'_{:, \dots, :, k_1, \dots, k_N} \in R^{I_1 \times I_2 \dots \times I_N}$, and the local feature of this block can be extract as follows:

$$\mathcal{H}_{:, \dots, :, k_1, \dots, k_N}^{(1)} = \mathcal{X}'_{:, \dots, :, k_1, \dots, k_N} \times_1 U_1^{(1)} \times \dots \times_N U_N^{(1)} + \mathcal{E}^{(1)} \quad (9)$$

$$\mathcal{H}^{(1)} = \mathcal{X}' \times_1 U_1^{(1)} \times \dots \times_N U_N^{(1)} \oplus \mathcal{E}^{(1)}. \quad (10)$$

Then, the features of arbitrary data blocks of the first hidden layer could be extracted by Equation (9). For a fixed data block, the above operation is also equivalent to multiple convolution operations on the data block, and the number of convolution kernels is $J_1 J_2 \dots J_N$. After learning the low-dimensional features of each data block through Equation (10), we further obtain the interactive information between the blocks through Equation (11). For simplicity, we denote $(k_1 + \sum_{n=2}^N (k_n - 1) \prod_{t=1}^{n-1} k_t)$ -th feature blocks as $\mathcal{H}_{k_1, \dots, k_N}^{(1)} \in R^{J_1 \times J_2 \dots \times J_N}$, $1 \leq j_n \leq J_n$, then we have

$$\left(\mathcal{H}^{(1)} \times_{N+1} V_{N+1}^{(1)} \right)_{l_1, j_2, \dots, j_N} = \sum_{j_1=1}^{J_1} \mathcal{H}_{j_1, \dots, j_N}^{(1)} \times V_{N+1(l_1, j_1)}^{(1)} \quad (11)$$

$$\mathcal{Z}^{(1)} = h \left\{ \mathcal{H}^{(1)} \times_{N+1} V_{N+1}^{(1)} \times_{N+2} V_{N+2}^{(1)} \dots \times_{2N} V_{N+2}^{(1)} \right\}. \quad (12)$$

In fact, Equation (11) is equivalent to combining the features of the data block along the mode-($N+1$). Similarly, we can continue to combine feature blocks along the remaining orders and generate multiple global features by Equation (12), and the overall process is shown in Figure 4.

Fig. 4. The structure of multi-layer QTNN.

4.3 Forward Propagation of QTNN

In this section, we formally define the structure of the QTNN. Without loss of generality, we can regard TNN as a special case of QTNs when the input data is not processed by convolution operations. In addition, we abstract the structure of QTNN into three layers, namely, input layer, hidden layer, and output layer, and corresponding forward computation process is also provided.

Input layer: This layer mainly includes tensor convolution, feature extraction, and feature interaction. Considering a high-order input tensor $\mathcal{X} \in R^{I_1 \times I_2 \times \dots \times I_N}$ and corresponding convolution feature tensor is $\mathcal{X}' \in R^{I_1 \times \dots \times I_N \times k_1 \times \dots \times k_N}$. Obviously, QTNN will degenerate into TNN when the convolution operations is not performed. First, we can get the low-dimensional feature of each data block in the first layer by

$$\mathcal{H}^{(1)} = \mathcal{X}' \times_1 U_1^{(1)} \times \dots \times_N U_N^{(1)} \oplus \mathcal{E}^{(1)}, \quad (13)$$

where $U_n^{(1)} \in R^{J_n \times I_n}$, $1 \leq n \leq N$. Actually, Equation (13) is also equivalent to convolution operation of CNN; the difference between two models is that the convolution kernel used here is composed of a rank-1 tensor. In addition, each data block is equivalent to being indirectly pooled, and its size is changed from the original $I_1 \times I_2 \times \dots \times I_N$ to $J_1 \times J_2 \times \dots \times J_N$. After obtaining the feature of data blocks of original data, feature blocks can be further interacted in the following way:

$$\mathcal{A}^{(1)} = \mathcal{H}^{(1)} \times_{N+1} V_{N+1}^{(1)} \times \dots \times_{2N} V_{2N}^{(1)}, \quad (14)$$

where $V_{N+n}^{(1)} \in R^{s_n \times k_n}$, $1 \leq n \leq N$, then the output feature tensor is $\mathcal{Z}^{(1)} = h\{\mathcal{A}^{(1)}\} \in R^{J_1 \times \dots \times J_N \times s_1 \times \dots \times s_N}$. Actually, $\mathcal{Z}^{(1)}$ is composed of $s_1 s_2 \dots s_N$ feature blocks, and each block can be further regarded as a compressed feature of $\mathcal{X}$.

Middle layer: For the l th layer, $2 \leq l \leq L-1$, the $2N$ -order tensors $\mathcal{Z}^{(l-1)} \in R^{J_1^{l-1} \times \dots \times J_N^{l-1} \times s_1^{l-1} \times \dots \times s_N^{l-1}}$ and $\mathcal{Z}^{(l)} \in R^{J_1^l \times \dots \times J_N^l \times s_1^l \times \dots \times s_N^l}$ are the output tensors of layer $l-1$ and layer l , respectively, and the relationship between them is given by the following equations:

$$\mathcal{H}^{(l)} = \mathcal{Z}^{(l-1)} \times_1 U_1^{(l)} \times \dots \times_N U_N^{(l)} \oplus \mathcal{E}^{(l)}, \quad (15)$$

$$\mathcal{A}^{(l)} = \mathcal{H}^{(l)} \times_{N+1} V_{N+1}^{(l)} \times \dots \times_{2N} V_{2N}^{(l)}, \quad (16)$$

$$\mathcal{Z}^{(l)} = h\{\mathcal{A}^{(l)}\}, \quad (17)$$

where $U_n^{(l)} \in R^{J_n^l \times J_n^{l-1}}$, $V_n^{(l)} \in R^{s_n^l \times s_n^{l-1}}$, and $\mathcal{E}^{(l)} \in R^{J_1^l \times \dots \times J_N^l}$ is bias tensor.

Output layer: For the classification task of C classes, we use the softmax function $\text{softmax}(\cdot)$ and the tensor contraction operation to output the probability of each class. The specific definition of layer L is given as follows:

$$\mathbf{a}^{(L)} = \langle \mathcal{W}, \mathcal{Z}^{(L-1)} \rangle_c + \mathbf{b}, \quad (18)$$

$$a_c^{(L)} = \langle \mathcal{W}_{:, \dots, :, c}, \mathcal{Z}^{(L-1)} \rangle + b_c, 1 \leq c \leq C, \quad (19)$$

$$\mathbf{y} = \text{softmax}(\mathbf{a}^{(L)}) = \frac{\exp(\mathbf{a}^{(L)})}{\sum_{i=1}^C \exp(a_i^{(L)})}, \quad (20)$$

where $\mathbf{y}, \mathbf{b} \in R^C$ correspond to the probability distribution of the output label and bias vector, respectively, and $\mathcal{W} \in R^{J_1^{L-1} \times \dots \times J_N^{L-1} \times s_1^{L-1} \times \dots \times s_N^{L-1} \times C}$ is $2N + 1$ -order parameter tensor, which is used to output the probability of each class, and $\mathcal{W}_{:, \dots, :, c}$ is $2N$ -order tensor, which is obtained by setting the index of the $(2N + 1)$ -th order to c .

Classification loss function: Given a set of high-order data tensors, and the corresponding class labels $\{\mathcal{X}_k, r^k\}$, $1 \leq k \leq K$, $y^k = \{y_c^k\}_{c=1}^C$ is the output probability distribution of the network. Then, the classification cross-entropy loss function is defined as follows:

$$\text{Loss}(\Theta) = \sum_{k=1}^K E_k(\Theta) = - \sum_{k=1}^K \sum_{c=1}^C r_c^k \ln y_c^k, \quad (21)$$

where $\Theta = \{U_n^{(l)}, V_{N+n}^{(l)}, \mathcal{E}^{(l)}, \mathcal{W}, 1 \leq n \leq N, 1 \leq l \leq L - 1\}$ represents the parameters of the network, and the label vector r^k uses one-hot coding method:

$$r_c^k = \begin{cases} 1, & \mathcal{X}_k \text{ belongs to } c\text{th class} \\ 0, & \text{otherwise.} \end{cases} \quad (22)$$

4.4 Back-propagation of QTNN

In this section, our main work is to minimize the classification loss function through the gradient optimization method and further provide a high-order back-propagation algorithm for QTNN. The most important step in this process is to calculate the gradient of the loss function with respect to the network parameters $\Theta = \{U_n^{(l)}, V_{N+n}^{(l)}, \mathcal{E}^{(l)}, \mathcal{W}, 1 \leq n \leq N, 1 \leq l \leq L - 1\}$. To solve this problem, we derive the following parameter update formulas:

(1) **Output layer:** For the output of the last layer of the network $a^{(L)}$, we can compute the gradient of $E_k(\Theta)$ with respect to each element of $a^{(L)}$ as follows:

$$d_c^{(L)} \doteq \frac{\partial E_k(\Theta)}{\partial a_c^{(L)}} = \sum_{t=1}^C \frac{\partial E_k(\Theta)}{\partial y_t^k} \frac{\partial y_t^k}{\partial a_c^{(L)}} = y_c^k - a_c^{(L)}. \quad (23)$$

For simplicity, we can denote the above formula as:

$$d^{(L)} \doteq \frac{\partial E_k(\Theta)}{\partial a^{(L)}} = y^k - a^{(L)}, \quad (24)$$

then we can derive the derivative of the parameters of the last layer in the following way:

$$\frac{\partial E_k(\Theta)}{\partial \mathcal{W}_{k_1, \dots, l_N, c}} = \frac{\partial E_k(\Theta)}{\partial a_c^{(L)}} \frac{\partial a_c^{(L)}}{\partial \mathcal{W}_{k_1, \dots, l_N, c}} = d_c^{(L)} \mathcal{Z}_{k_1, \dots, k_N}^{(L-1)}, \quad (25)$$

$$\frac{\partial E_k(\Theta)}{\partial b_c} = \frac{\partial E_k(\Theta)}{\partial a_c^{(L)}} \frac{\partial a_c^{(L)}}{\partial b_c} = d_c^{(L)}, \quad (26)$$

then we can rewrite the above formulas in tensor form as:

$$\frac{\partial E_k(\Theta)}{\partial \mathcal{W}_{i,\dots,c}} = d_c^{(L)} \mathcal{Z}^{(L-1)}, \quad (27)$$

$$\frac{\partial E_k(\Theta)}{\partial b} = d^{(L)}. \quad (28)$$

After getting the gradient of the parameters of layer L , we can use the gradient descent algorithm to update the parameters of the L th layer. To recursively calculate the gradient of the parameters of layer l , $1 \leq l \leq L-1$, we continue to calculate the gradient of $\mathcal{Z}^{(L-1)}$. According to the Equation (18), we have

$$\frac{\partial E_k(\Theta)}{\partial \mathcal{Z}_{k_1,\dots,k_N}^{(L-1)}} = \sum_{t=1}^C \frac{\partial E_k(\Theta)}{\partial a_t^{(L)}} \frac{\partial a_t^{(L)}}{\partial \mathcal{Z}_{k_1,\dots,k_N}^{(L-1)}} = \sum_{t=1}^C d_t^{(L)} \mathcal{W}_{k_1,\dots,k_N,t}, \quad (29)$$

$$\frac{\partial E_k(\Theta)}{\partial \mathcal{A}_{k_1,\dots,k_N}^{(L-1)}} = \frac{\partial E_k(\Theta)}{\partial \mathcal{Z}_{k_1,\dots,k_N}^{(L-1)}} \times h' \{ \mathcal{A}_{k_1,\dots,k_N}^{(L-1)} \}. \quad (30)$$

For simplicity, we can also denote the above formula as:

$$\delta^{(L-1)} \doteq \frac{\partial E_k(\Theta)}{\partial \mathcal{A}^{(L-1)}} = (\mathcal{W} \times_{2N+1} d^{(L)}) * h' \{ \mathcal{A}^{(L-1)} \}, \quad (31)$$

where $*$ is Hadamard product.

(2) **Hidden layer:** For the l th layer, $1 \leq l \leq L-1$, we can compute the gradients of $\Theta = \{U_i^{(l)}, V_{N+i}^{(l)}, \mathcal{E}^{(l)}, 1 \leq i \leq N\}$ through the following formula:

$$\begin{aligned} \frac{\partial E_k(\Theta)}{\partial V_{i(m,n)}^{(l)}} &= \sum_{j_1=1}^{J_1^l} \cdots \sum_{j_N=1}^{J_N^l} \sum_{s_1=1}^{s_1^l} \cdots \sum_{s_N=1}^{s_N^l} \frac{\partial E_k(\Theta)}{\partial \mathcal{A}_{j_1 \cdots j_N, s_1 \cdots s_N}^{(l)}} \frac{\partial \mathcal{A}_{j_1 \cdots j_N, s_1 \cdots s_N}^{(l)}}{\partial V_{i(m,n)}^{(l)}} \\ &= \sum_{j_1=1}^{J_1^l} \cdots \sum_{j_{i-1}=1}^{J_{i-1}^l} \sum_{j_{i+1}=1}^{J_{i+1}^l} \cdots \sum_{s_N=1}^{s_N^l} \delta_{j_1 \cdots j_N, s_1 \cdots s_N}^{(l)} \\ &\quad \times \left(\sum_{t_1=1}^{s_1^l} \cdots \sum_{t_{i-1}=1}^{s_{i-1}^l} \sum_{t_{i+1}=1}^{s_{i+1}^l} \cdots \sum_{t_N=1}^{s_N^l} \mathcal{H}_{j_1 \cdots j_N, t_1 \cdots t_N}^{(l-1)} \times V_{1(s_1, t_1)}^{(l)} \cdots \times V_{2N(s_N, t_N)}^{(l)} \right). \end{aligned} \quad (32)$$

For simplicity, we can denote Equation (32) as follows:

$$\frac{\partial E_t}{\partial V_{i(m,n)}^{(l)}} = \langle \delta_{j_i=m, \mathcal{T}_{j_i=n}}^{(l)}, \mathcal{T}_{j_i=n} \rangle = \mathcal{H}_{j_i=n}^{(l-1)} \times_{N+1} V_1^{(l)} \cdots \times_{N+i-1} V_{i-1}^{(l)} \times_{N+i+1} V_{i+1}^{(l)} \cdots \times_{2N} V_{2N}^{(l)}. \quad (33)$$

As for the gradient of $\mathcal{H}^{(l)}$, for simplicity, we can also denote it as $\beta^{(l)} \doteq \frac{\partial E_k(\Theta)}{\partial \mathcal{H}^{(l)}}$, then we have:

$$\frac{\partial E_k(\Theta)}{\partial \mathcal{H}_{j_1, \dots, j_N, s_1, \dots, s_N}^{(l)}} = \sum_{t_1=1}^{s_1^l} \cdots \sum_{t_N=1}^{s_N^l} \delta_{j_1, \dots, j_N, t_1, \dots, t_N}^{(l)} \times V_{N+1(t_1, s_1)}^{(l)} \cdots \times V_{2N(t_N, s_N)}^{(l)}. \quad (34)$$

Similarly, we can update the factor matrix $U_i^{(l)}$, $1 \leq i \leq N$ by the following formula:

$$\frac{\partial E_t}{\partial U_{i(m,n)}^{(l)}} = \langle \beta_{j_i=m}^{(l)}, \mathcal{T}_{j_i=n} \rangle = \mathcal{Z}_{j_i=n}^{(l-1)} \times_1 U_1^{(l)} \cdots \times_{i-1} U_{i-1}^{(l)} \times_{i+1} U_{i+1}^{(l)} \cdots \times_N U_N^{(l)}. \quad (35)$$

As for the gradients of bias tensor in layer l , we have:

$$\frac{\partial E_k(\Theta)}{\partial \mathcal{E}_{j_1, \dots, j_N}^{(l)}} = \sum_{s_1=1}^{s_1^l} \dots \sum_{s_N=1}^{s_N^l} \frac{\partial E_k(\Theta)}{\partial \mathcal{H}_{j_1, \dots, j_N, s_1, \dots, s_N}^{(l)}}. \quad (36)$$

Considering that the bias tensor increases exponentially with the increase of the order, we can use the CP model to generate bias indirectly when the dimension of the input data is extremely high. The gradient of the corresponding parameter is given below.

Assuming $\mathcal{E}^{(l)} = [B^{(1)}, \dots, B^{(N)}] = \sum_{k=1}^K b_k^{(1)} \circ b_k^{(2)} \dots \circ b_k^{(N)}$, and $B^{(n)} = [b_1^{(n)}, \dots, b_K^{(n)}] \in R^{I_n \times K}$, $1 \leq n \leq N$. The mode- n unfolding of $\mathcal{E}^{(l)}$ is given as follows:

$$\mathcal{E}_{(n)}^{(l)} = \text{unfolding}(\mathcal{E}^{(l)}, n) = B^{(n)}(B^{(n+1)} \dots \odot B^{(N)} \dots \odot B^{(1)})^T, \quad (37)$$

where $\odot$ is defined as Khatri-Rao product. Then, we have

$$\frac{\partial \mathcal{E}_{(n)}^{(l)}}{\partial B^{(n)}} = \frac{\partial B^{(n)}(B^{(n+1)} \dots \odot B^{(N)} \dots \odot B^{(1)})^T}{\partial B^{(n)}}. \quad (38)$$

In Algorithm 1, a high-order error back-propagation algorithm of tensor neural network is provided, which realizes the transformation of traditional back-propagation algorithm from vector space to higher-order tensor space.

ALGORITHM 1: High-order Error Back-propagation Algorithm

Input: Original high-dimensional dataset and corresponding class labels $\{\mathcal{X}^{(k)}, r^k\}$, $\mathcal{X}^{(k)} \in R^{I_1 \times I_2 \dots \times I_N}$, $1 \leq k \leq K$.

Output: The parameter of tensor neural network: $\Theta = \{U_n^{(l)}, V_{N+n}^{(l)}, \mathcal{E}^{(l)}, \mathcal{W}, 1 \leq n \leq N, 1 \leq l \leq L-1\}$.

1 **begin**

2 Random initialization for all the parameters: Θ .

3 **for** $epoch = 1$ to N_epoch **do**

4 Feed_forward the QTNN through defined structure

5 Using Equations (15)–(18), (20), and (21) to compute $\mathcal{H}^{(l)}$, $\mathcal{A}^{(l)}$ and $\mathcal{Z}^{(l)}$, $1 \leq l \leq L-1$.

6 Evaluate $d^{(L)}$ through Equation (24) and back-propagate to the previous layers using Equations (31) and (34).

7 Update $\Theta = \{U_n^{(l)}, V_{N+n}^{(l)}, \mathcal{E}^{(l)}, \mathcal{W}, 1 \leq n \leq N\}$ by the gradients in Equations (27), (28), (33), (35), and (36).

8 Repeat the process until the maximum number of iterations is reached or the parameters Θ converged.

9 **return** the parameter: $\Theta = \{U_n^{(l)}, V_{N+n}^{(l)}, \mathcal{E}^{(l)}, \mathcal{W}, 1 \leq n \leq N, 1 \leq l \leq L-1\}$.

5 MODEL ANALYSIS

Considering an N -order input data tensor $\mathcal{X} \in R^{I_1 \times I_2 \dots \times I_N}$, and the corresponding compressed rank of each order is $R_n, R_n \leq I_n, 1 \leq n \leq N$. In fact, the tensor neural networks directly takes N -dimensional tensor as an input to avoid the vectorization of data, which also preserves the structural information of the original data well. In addition, the weight $W \in R^{R_1 R_2 \dots R_N \times I_1 I_2 \dots I_N}$ generated by factor matrices $U^{(n)}, 1 \leq n \leq N$, has fewer parameters when compared to the standard fully connected layer. More specifically, the standard fully connected layer taking a flatten vector as input, which has n_{FC} parameters,

$$n_{FC} = \prod_{n=1}^N I_n R_n + \prod_{n=1}^N R_n, \quad (39)$$

where $\prod_{n=1}^N I_n R_n$, $\prod_{n=1}^N R_n$ are the size of weight matrix and bias vector in fully connected layer, respectively. As for the nonlinear Tucker layer, the hidden feature is obtained by multi-linear transformations of the original data and N factor matrices, and it has n_{TL} parameters,

$$n_{TL} = \sum_{n=1}^N I_n R_n + \prod_{n=1}^N R_n, \quad (40)$$

where $\sum_{n=1}^N I_n R_n$ is the number of parameters in factor matrices, and $\prod_{n=1}^N R_n$ corresponds to the number of parameters in bias tensor. Actually, we usually use CP model to generate bias to avoid the size of the bias tensor increasing exponentially with the orders. Assuming $\mathcal{E}^{(l)} = [B^{(1)}, \dots, B^{(N)}] = \sum_{k=1}^K b_k^{(1)} \circ b_k^{(2)} \dots \circ b_k^{(N)}$, and $B^{(n)} = [b_1^{(n)}, \dots, b_K^{(n)}] \in R^{R_n \times K}$, $1 \leq n \leq N$, then we have

$$n_{TL} = \sum_{n=1}^N I_n R_n + N R_n K. \quad (41)$$

To further improve the learning ability of tensor neural network, we introduce multiple high-order convolution layers in front of TNN network to extract local features of high-order tensor. Assuming there are three convolution layers, the corresponding number of convolution kernels is n_1, n_2, n_3 , respectively, and the size of all the convolution kernels is $k \times k \dots \times k$, then the number of parameters in the convolution layers is about n_{CL} ,

$$n_{CL} = (n_1 + n_2 n_1 + n_3 n_2) k^N. \quad (42)$$

6 EXPERIMENTS

To verify the performance of tensor neural networks in the classification tasks, we conducted experiments on three representative datasets, namely, MNIST, CIFAR10, and CIFAR100 datasets. We take a unified computer resource, and the system is Ubuntu16.04LTS and the hardware is Intel+2 X RTX 2080Ti. The experimental computer is equipped with a CPU of Intel Core TM i7-6700 @3.2 GHz with eight cores and a GPU of NVIDIA GeForce GTX TITAN and a memory of 32 G RAM. CUDA 10.0 is used for parallel acceleration of experimental codes. The experiments are implemented based on PyTorch framework, and the tensor models are calculated by Tensorly library. All parameters of the proposed models are obtained through initial training, and the image classification models are obtained by training 200 epochs. The initial learning rate is set as 0.003, and the SGD optimizer is used in the experiments, and the learning rate momentum is 0.9. For MNIST dataset, the batch size is set to 16 for training, and the CIFAR10 and CIFAR100 are set to 128.

6.1 The Experiment on the MNIST Dataset

First, we estimate the performance of the proposed models on the MNIST dataset, which is compiled by the NIST, consists of 70,000 grayscale (0–255) images of 10 handwritten digits (0–9), and each image in the MNIST dataset consists of $28 * 28$ pixels. The training set has 60,000 handwritten digital pictures, and the test set has 10,000 handwritten digital pictures. The classification accuracy of the test set is used as the evaluation basis of our network.

In the first experiment, we designed a variety of different network topologies for each model, namely, NN, TNN, CNN, and QTNN, and compared the performance of different types of networks. Two-layer and three-layer **tensor neural networks (TNNs)** and **quantized tensor neural networks (QTNNs)** are compared with classic NNs and CNNs, respectively. A two-layer NN model with a vector of length 784 as input, 70 hidden features in the middle layer, 10 outputs for the classification, is simply denoted as (784-70-10). In addition, a two-layer tensor network with matrix of size $28 * 28$ as input, and the size of compressed feature in the hidden layer is $10 * 10$, can be

Table 1. Comparison of Classification Accuracy (%) and the Number of Parameters of Different Networks on MNIST

Model	Topology	Test accuracy	Par. size
NN	784-70-10	96.7 ± 0.1	55,660
NN	784-300-10	96.9 ± 0.2	238,510
NN	784-300-300-10	96.8 ± 0.2	328,810
TNN	28^*28-10^*10-10	96.3 ± 0.2	1,670
TNN	$28^*28-16^*16-10^*10-10$	97.4 ± 0.3	2,582
CNN	5×5 , 25 maps, maxpooling, 1 layer, 4,900-10	98.4 ± 0.1	49,660
CNN	5×5 , 25 maps, maxpooling, 1 layer, 4,900-512-10	98.9 ± 0.2	2,561,172
QTNN	1 convolution layer, $5^*5^*28^*28-3^*3^*5^*5-10$	98.4 ± 0.3	3,445
QTNN	1 convolution layer, $5^*5^*28^*28-5^*5^*10^*10-3^*3^*5^*5-10$	98.7 ± 0.2	6,375

denoted as (28^*28-10^*10-10) . As for the multi-layer TNN models, which is composed by multiple nonlinear Tucker layers, it can achieve better results than the traditional NN model while using fewer parameters. This is mainly because we use tensor models to efficiently process multidimensional data without destroying the structural information of the original data. Although we add multiple tensor layers to the TNN models, the number of parameters in each layer is almost negligible compared with the parameters in the full connection layer in NNs. In the implementation of QTNNs and classic CNNs, we use one convolution layer to extract the local feature of original data. More specifically, the first layer is composed of 25 convolution kernels with the size of 5×5 , the stride of convolution is set as 1, and the padding is also adopted to preserve the size of the original data.

As shown in Table 1, we use multiple nonlinear Tucker layers in the TNN model to gradually extract global features of original data, and each layer is actually composed of two parameter factor matrices. Obviously, different from adding a traditional fully connected layer to NNs, which easily brings an unbearable amount of parameters, adding a nonlinear Tucker layer in TNNs or QTNNs hardly changes the amount of parameters of the original models. Moreover, it can also achieve better classification accuracy than NN models when using multi-layer network structure. As mentioned earlier, this also further demonstrates that TNN models have the ability to learn potential spatial structure information from observed data. To make better use of the local information of the data, the convolution operations are integrated into the TNNs to extract the high-order neighborhood features of the data, and further forms the QTNN network, which well combines the advantages of CNNs and TNNs. As for QTNN models, we mainly extract the neighborhood features of each element through a layer of 5^*5 convolution, and then directly input high-order feature tensor into the TNNs instead of pooling. In fact, the pooling operation is indirectly completed by the subsequent nonlinear Tucker layer. Different from the huge amount of parameters brought by multiple fully connected layers in CNNs, QTNNs can more effectively improve the classification accuracy at the cost of a small amount of parameters by adding a nonlinear Tucker layer. QTNNs can also achieve almost the same performance as CNNs, and it is significantly better than NNs and TNNs.

6.2 The Experiment on the CIFAR-10 Dataset

CIFAR-10 is a commonly used dataset for universal object classification, which includes 10 different types of objects, such as vehicles and animals. The data of CIFAR-10 are all three-channel RGB images, which can be represented as 3-order tensors with the size of $3 \times 32 \times 32$. All the samples of CIFAR10 are from the real world, and the size and background of the objects are significantly

Table 2. Comparison of Classification Accuracy (%) and the Number of Parameters of Different Networks on CIFAR-10

Model	Topology	Test accuracy	Par. size
NN	3,072-500-10	61.0 ± 0.3	1,541,510
NN	3,072-1,000-10	61.8 ± 0.2	3,083,010
NN	3,072-300-300-10	63.3 ± 0.3	1,792,010
TNN	$3*32*32-(25*32*32, 5*5*32*32)-3*3*8*8-10$	65.0 ± 0.5	9,523
TNN	$3*32*32-(25*32*32, 5*5*32*32)-5*5*16*16-3*3*8*8-10$	65.8 ± 0.2	10,597
CNN	$3 \times 5 \times 5$, 25 maps, maxpooling, 1 layer, 6400-10	64.2 ± 0.3	64,710
CNN	$3 \times 5 \times 5$, 25 maps, maxpooling, 1 layers, 6400-512-10	74.3 ± 0.3	3,283,143
QTNN	1 convolution layer, $5*5*32*32-3*3*8*8-10$	69.3 ± 0.3	7,076
QTNN	1 convolution layer, $5*5*32*32-5*5*10*10-3*3*8*8-10$	71.5 ± 0.4	7,514

different, which also puts forward higher requirements for the performance of the classification network.

Table 2 shows the classification results and the number of parameters of related models, including NNs, CNNs, TNNs, and QTNNs. For different network topologies, the classification performance of TNNs is significantly better than the NNs. This also further shows that preserving the structural information of the original data is very beneficial and necessary for the construction of neural networks. As mentioned above, TNN models are more inclined to analyze the overall structural properties of original data and ignore the local neighborhood features. However, the local features of the data are of great significance for the analysis of images, videos, and other data, which is the main reason for the poor classification performance of TNN models compared with QTNN models. In addition, we adopt a single convolution layer with 25 convolution kernels to extract the local feature of original data, and the size of each kernel is $3 \times 5 \times 5$. The classification accuracy of CNN with multiple fully connected layers is compared to QTNN with multiple tensor layers, and it is clear that the classification accuracy of QTNNs is significantly better than CNNs with only one fully connected layer. When we introduce multiple full connection layers into CNNs, although the classification accuracy can be significantly improved, it will also bring a huge number of parameters, which is extremely inefficient compared with the way to improve the model performance by introducing multiple tensor layers into QTNNs.

To fully explain the local feature of the data is more conducive to improve the classification accuracy of the different models, we use three convolution layers in CNNs and QTNNs to expand the local field of view of the convolution features. The numbers of convolution kernels of three different layers are 16, 32, 64, respectively, the sizes of the corresponding convolution kernel are $3 \times 5 \times 5$, $16 \times 5 \times 5$, $32 \times 5 \times 5$. We also compare the performance of CNNs and QTNNs with a single convolutional layer, and the first layer of corresponding networks are constructed by 64 convolutional kernels of size $3 \times 5 \times 5$. For the QTNNs, we do not adopt pooling operation after each convolution operation, which also guarantees the integrity of the structural information of the original data. As shown in Table 3, the performance of the two kinds of networks is significantly improved with the increase of the number of convolutional layers. Moreover, the classification accuracy of QTNNs is generally better than that of CNNs.

6.3 The Experiments of Deep Tensor Neural Networks on the CIFAR-10 and CIFAR-100

To meet the needs of industrial and living applications, we further integrate the QTNN network into the deep neural network VGG model and build a deeper tensor network to improve the accuracy of classification. We evaluate the performance of related models on two datasets, i.e.,

Table 3. Classification Accuracy (%) of Various Models with Different Number of Convolution Layers on CIFAR-10

Model	Topology	Test accuracy	Par.size
CNN	1 <i>Conv_layer</i> , $3 \times 5 \times 5$, 64 <i>maps</i> , <i>maxpooling</i> , 6,400-10	66.5 ± 0.3	165,642
CNN	3 <i>Conv_layers</i> (16, 32, 64), <i>maxpooling</i> , 1,024-10	80.6 ± 0.3	33,834
QTNN	1 <i>Conv_layer</i> , $8^*8^*32^*32-4^*4^*8^*8-10$	71.0 ± 0.4	12,682
QTNN	1 <i>Conv_layer</i> , $8^*8^*32^*32-8^*8^*16^*16-4^*4^*8^*8-10$	73.5 ± 0.3	13,198
QTNN	3 <i>Conv_layers</i> (16, 32, 64), $8^*8^*32^*32-4^*4^*8^*8-10$	80.8 ± 0.2	34,474
QTNN	3 <i>Conv_layers</i> (16, 32, 64), $8^*8^*32^*32-8^*8^*16^*16-4^*4^*8^*8-10$	81.5 ± 0.2	35,626

Fig. 5. The structure of deep tensor neural networks.

CIFAR-10, CIFAR-100. CIFAR-100 dataset is more complex compared with CIFAR-10, and it contains 100 subclasses, and each subclass contains 600 images, including 500 training images and 100 test images. The 100 classes are grouped into 20 broad categories. Each image carries a small “fine” label and a large “coarse” label. The experimental result of VGG16 is used as a reference for subsequent experiments. In this experiment of deep classification network, we introduce BN layer to avoid the gradual deviation or change of the distribution of activation input values during the training process as the network deepens, forcing its distribution back to the standard normal distribution with a mean of 0 and a variance of 1.

To construct a deeper network model similar to the VGG structure more intuitively, we can broadly consider that deep tensor neural network consists of two basic layers, namely, convolution layer and tensor layer. The convolution layer is used to extract the local features of original data, and the tensor layer is used to further analyze the features while maintaining the data structure. Then, we construct two different network topologies based on the two basic layers, namely, topology a and topology b.

As shown in Figure 5, the first structure mainly uses the convolution operation to extract the features of the data without introducing the pooling operation to change the size of the original data, and the pooling operation is indirectly completed by the subsequent tensor layers. The second structure is to reduce the size of the data by introducing pooling operation while convolution is adopted to extract data features. The classification accuracy of three models is summarized in

Table 4. Classification Accuracy (%) of Deep Tensor Neural Networks

Dataset	Model	Test accuracy (%)	Par. size
CIFAR-10	VGG-16	91.0 ± 0.2	16,392,100
	QTNN-a	88.0 ± 0.6	1,151,882
	QTNN-b	90.4 ± 0.3	1,148,874
CIFAR-100	VGG-16	67.5 ± 0.2	16,392,100
	QTNN-a	66.5 ± 0.3	1,151,882
	QTNN-b	66.8 ± 0.2	1,148,874

Table 4, and it can be clearly seen that in the construction of deep network, the pooling operation is helpful to improve the classification accuracy of the model and reduce the time of parameter optimization. Actually, the parameters of the deep tensor neural network are mainly derived from the convolutional layers, and the parameters of the tensor layers can be almost ignored, which is markedly different from the CNNs and NNs. In addition, we try to make the features obtained by convolution operations have the same spatial structure as the original data through higher-order quantization operations and learn more compact neighborhood features to avoid redundancy. Although the classification accuracy of VGG is slightly higher than that of tensor neural networks, VGG adopts a more complex network structure, including the introduction of more convolution layers to extract features and more full connection layers with a large number of parameters, which indirectly verifies the high efficiency of tensor neural networks.

7 CONCLUSION

In this article, we propose a **quantized tensor neural network (QTNN)**, which effectively integrates the simplicity of tensor networks and the powerful learning ability of neural networks. Different from the traditional tensor neural network models, which tend to analyze the overall structural information of the data and ignore the local structural features of the data, QTNN solves this problem effectively by further introducing the convolution operations and realizing the efficient processing of the data while maintaining the structure of the original data. Moreover, we make the features obtained by convolution operations have the same spatial structure as the original data through higher-order quantization operations to learn more compact neighborhood features to avoid redundancy. We also develop a high-order error back-propagation algorithm based on tensor networks for parameter optimization. Experimental results on three representative datasets further demonstrate the effectiveness and simplicity of our proposed models.

By combining tensor network with neural network, the models proposed in this article have achieved good performance on classification task. In future work, we will continue our research in the following directions: (1) Establishing the potential correlation of hidden layer features of deep neural network and reducing redundant features by tensor network. (2) Based on the complete theory of multi-linear algebra, further exploring various tensor neural network models with stronger interpretation.

REFERENCES

- [1] Evrim Acar, Morten A. Rasmussen, Francesco Savorani, Tormod Næs, and Rasmus Bro. 2013. Understanding data fusion within the framework of coupled matrix and tensor factorizations. *Chemomet. Intell. Lab. Syst.* 129 (2013), 53–63.
- [2] Giuseppe G. Calvi, Ahmad Moniri, Mahmoud Mahfouz, Qibin Zhao, and Danilo P. Mandic. 2019. Compression and interpretability of deep neural networks via Tucker tensor layer: From first principles to tensor valued back-propagation. *arXiv preprint arXiv:1903.06133* (2019).

- [3] Xiaochun Cao, Xingxing Wei, Yahong Han, and Dongdai Lin. 2014. Robust face clustering via tensor decomposition. *IEEE Trans. Cyber.* 45, 11 (2014), 2546–2557.
- [4] Jen-Tzung Chien and Yi-Ting Bao. 2017. Tensor factorized neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 29, 5 (2017), 1998–2011.
- [5] Nadav Cohen, Or Sharir, and Amnon Shashua. 2016. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*.
- [6] Beyza Ermiş, Evrim Acar, and A. Taylan Cemgil. 2015. Link prediction in heterogeneous data via generalized coupled tensor factorization. *Data Mining Knowl. Discov.* 29, 1 (2015), 203–236.
- [7] Lars Grasedyck. 2010. Hierarchical singular value decomposition of tensors. *SIAM J. Matrix Anal. Applic.* 31 (2010), 2029–2054. DOI: <https://doi.org/10.1137/090764189>
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.
- [9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).
- [10] Tamara Kolda and Brett Bader. 2009. Tensor decompositions and applications. *SIAM Rev.* 51 (2009), 455–500.
- [11] Jean Kossaifi, Zachary C. Lipton, Arinbjörn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. 2020. Tensor regression networks. *J. Mach. Learn. Res.* 21 (2020), 1–21.
- [12] Liwei Kuang, Fei Hao, Laurence T. Yang, Man Lin, Changqing Luo, and Geyong Min. 2014. A tensor-based approach for big data representation and dimensionality reduction. *IEEE Trans. Emerg. Topics Comput.* 2 (2014), 280–291.
- [13] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. 2014. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint arXiv:1412.6553* (2014).
- [14] Yang Liu, Yan Liu, Shenghua Zhong, and Keith Chan. 2012. Tensor distance based multilinear globality preserving embedding: A unified tensor based dimensionality reduction framework for image and video classification. *Expert Syst. Applic.* 39 (2012), 10500–10511.
- [15] Alexander Novikov, Dmitry Podoprikin, Anton Osokin, and Dmitr Vetrov. 2015. Tensorizing neural networks. *arXiv preprint arXiv:1509.06569* (2015).
- [16] Ivan V. Oseledets. 2011. Tensor-train decomposition. *SIAM J. Sci. Comput.* 33, 5 (2011), 2295–2317.
- [17] Anh-Huy Phan and Andrzej Cichocki. 2010. Tensor decompositions for feature extraction and classification of high dimensional datasets. *Nonlin. Theor. Applic.* 1 (2010), 37–68.
- [18] Surya Prasath, Rengarajan Pelapur, Guna Seetharaman, and Kannappan Palaniappan. 2019. Multiscale structure tensor for improved feature extraction and image regularization. *IEEE Trans. Image Process.* 28 (2019), 6198–6210.
- [19] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
- [20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*.
- [21] Puming Wang, Laurence T. Yang, Jintao Li, Jinjun Chen, and Shangqing Hu. 2019. Data fusion in cyber-physical-social systems: State-of-the-art and perspectives. *Inf. Fusion* 51 (2019), 42–57.
- [22] Yinchong Yang, Denis Krompass, and Volker Tresp. 2017. Tensor-train recurrent neural networks for video classification. In *International Conference on Machine Learning*. 3891–3900.
- [23] Qingchen Zhang, Laurence T. Yang, and Zhikui Chen. 2015. Deep computation model for unsupervised feature learning on big data. *IEEE Trans. Serv. Comput.* 9, 1 (2015), 161–171.
- [24] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *IEEE Conference on Computer Vision and Pattern Recognition*.
- [25] Yaliang Zhao, Laurence T. Yang, and Ronghao Zhang. 2017. A tensor-based multiple clustering approach with its applications in automation systems. *IEEE Trans. Industr. Inform.* 14 (2017), 283–291.

Received March 2021; revised July 2021; accepted September 2021