

Tensor Networks Meet Neural Networks: A Survey

Maolin Wang*, Yu Pan*, Xiangli Yang, Guangxi Li, and Zenglin Xu

Abstract—Tensor networks (TNs) and neural networks (NNs) are two fundamental types of data modeling approaches. TNs have been proposed as a solution to the curse of dimensionality faced by large-scale tensors by converting an exponential number of dimensions to polynomial complexity. Thus, they have attracted many studies in the fields of quantum physics and machine learning. On the other hand, NNs are computing systems inspired by the biological NNs that constitute human brains. Recently, NNs and their variants have achieved outstanding performance in various applications, e.g., computer vision, natural language processing, and robotics research. Interestingly, although these two types of networks come from different observations, they are inextricably linked via the common intrinsic multilinearity structure underlying both TNs and NNs. Consequently, a significant number of intellectual sparks regarding combinations of TNs and NNs have burst out. The combinations described as “tensor networks meet neural networks” are termed tensorial neural networks (TNNs) in this paper. This survey introduces TNNs based on three aspects. 1) Network Compression. TNs can greatly reduce parameters in NNs and satisfy the idea of constructing effective NNs. 2) Information Fusion. TNs can naturally and effectively enhance NNs with their ability to model the interactions among multiple modalities, views, or sources of various data. 3) Quantum Circuit Simulation. TNs can assist in designing and simulating quantum neural networks (QNNs). This survey also investigates methods for improving TNNs, examines useful toolboxes for implementing TNNs, and attempts to document TNN development and highlight its potential future directions. To the best of our knowledge, this is the first comprehensive survey to bridge the connections among NNs, TNs, and quantum circuits. We provide a curated list of TNNs at <https://github.com/tnbar/awesome-tensorial-neural-networks>.

Index Terms—Tensor Networks, Neural Networks, Network Compression, Information Fusion, Quantum Circuit Simulation

1 INTRODUCTION

TENSORS are high-order arrays that represent the multiway interactions among multiple modal sources. In contrast, vectors (i.e., first-order tensors) and matrices (i.e., second-order tensors) are accessed in only one or two modes, respectively. As a common data type, tensors have been widely observed in several scenarios [1], [2], [3], [4]. For instance, functional magnetic resonance imaging (fMRI) samples are inherently fourth-order tensors that are composed of three-dimensional voxels that change over time [5], [6], [7]. In quantum physics, variational wave functions used to study many-body quantum systems are also high-order tensors [8], [9]. For spatiotemporal traffic analysis, road flow/speed information, which is collected from multiple roads over several weeks, can also be structured as a third-order tensor (road segment \times day \times time of day) [10]. However, for higher-order tensors, when the number of modes increases, the total number of elements in the tensors increases exponentially, resulting in a catastrophe when storing and processing tensors. Such a phenomenon is also recognized as the “curse of dimensionality” [11].

Tensor Networks (TNs). TNs [8], [11], [12] are generally

countable collections of small-scale tensors that are interconnected by tensor contractions. These small-scale tensors are referred to as “components”, “blocks”, “factors”, or “cores”. Very large-scale tensors can be approximately represented in extremely compressed and distributed formats through TNs. Thus, it is feasible to implement distributed storage and efficient processing for high-order tensors that could not be dealt with before. By using TN methods, the curse of dimensionality can be alleviated or completely overcome [11]. Commonly used TN formats include CANDECOMP/PARAFAC (CP) [13], [14], [15], Tucker decomposition [16], [17], Block-term Tucker (BTT) decomposition [18], [19], [20], Matrix Product State (MPS)/Tensor Train (TT) decomposition [21], [22], [23], [24], Matrix Product Operators (MPO)/matrix Tensor Train (mTT) decomposition [21], [22], [23], [24], Tensor Ring (TR) decomposition [25], Tree TN/Hierarchical Tucker (HT) decomposition [26], Projected Entangled Pair State (PEPS)/Tensor Grid decomposition [8], [27], [28], Multiscale Entanglement Renormalization [29], etc. For the purpose of understanding the interconnected structures of TNs, a TN diagram was developed as a straightforward graphical diagram (which is discussed in Section 2.2). A TN can provide a theoretical and computational framework for the analysis of some computationally unacceptable tasks. For example, based on the low-rank structures of TNs, Pan et al. [30] were able to solve the quantum random circuit sampling problem in 15 hours using 512 graphics processing units (GPUs); this problem was previously believed to require over 10,000 years on the most powerful classic electronic supercomputer and effectively challenge the quantum supremacy of Google’s quantum computer called “Sycamore”. Other applications include brain analysis [31], quantum chemistry calculation [32], human face clustering [33], dimensionality reduction [34], missing value estimation [35], latent factor analysis [36], subspace learning [37], etc.

* Equal Contribution

- M. Wang was with the City University of Hong Kong, HKSAR, China.
E-mail: morin.w98@gmail.com
- Y. Pan was with the Harbin Institute of Technology Shenzhen, Shenzhen, China.
E-mail: iperryuu@gmail.com
- X. Yang was with the University of Electronic Science and Technology of China, Chengdu, China.
E-mail: xlyang@std.uestc.edu.cn
- G. Li was with the University of Technology Sydney, NSW, Australia.
E-mail: gxli2017@gmail.com
- Z. Xu was with the Harbin Institute of Technology Shenzhen, Shenzhen, China and the Pengcheng Laboratory, Shenzhen, China.
E-mail: zenglin@gmail.com

TABLE 1: An overview of TNNs and their utility. We first introduce the building of compact TNNs in different basic NN structures including CNNs, RNNs, Transformers, graph neural networks (GNNs) and RBMs in Section 3. Next, we explore the use of TNs in efficient information fusion methods based on tensor fusion and multimodal fusion in Section 4. Then, we discuss some applications involving TNs in quantum circuits and quantum TNNs in Section 5. Furthermore, we explain some training and implementation techniques for TNNs in Section 6. Finally, we introduce some general and powerful toolboxes for processing TNNs in Section 7.

Category	Subcategory	Detailed Models/Techniques	Section	
TNNs	Network Compression	CP-CNN [61], [62], [63], Tucker-CNN [64], [65], TT-CNN [66], [67], TR-CNN [68], BTT-CNN [69], TC-CNN [70], [71], HT-TT-CNN [72], T-Net [73], TTMT/TMT [74], PMT [75], CP-HOConv [76]	3.1	
		Recurrent Neural Networks	TT-RNN [77], BTT-RNN [69], [78], TR-RNN [79], HT-RNN [80], CP/Tucker-RNN [64], TC-RNN [70], [71] Kronecker-RNN [81], Tucker/CP/TT-RNN [82], HT-TT-RNN [72], KCP-RNN [83], Conv-TT-LSTM [84]	3.2
		Transformer	BTT-Transformer [85], MPO-Transformer [86], Tuformer [87], Tucker-Bert [88], Hypoformer [89]	3.3
		Graph Neural Networks	TGNNs [90], TGCN [91], DSTGNN [92]	3.4
		Restricted Boltzmann Machines	TT-RBM [93], TR-RBM [94], Tv-RBM [95], Mv-RBM [96]	3.5
	Information Fusion	Tensor Fusion Layers	TFL [90], [97], [98], LMF [99], PFN [100]	4.1
		Multimodal Pooling Layers	MUTAN [101], MCB [102], MLB [103] CIT [104]	4.2
	Quantum Circuit Simulation	Quantum Embedding	Image-Emb [105], Language-Emb [106], [107], [108]	5.1
		Quantum Data Processing	Supervised MPS [105], Tree-TN [109], Uniform-MPS [108], LPS [110]	5.2
		Quantum TNNs	ConvAC [107], [111], TLSM [112]	5.3
Utility of TNNs	Training Strategy	Stable Training	Mixed Precision [113], Yu Initialization [114]	6.1
		Rank Selection	PSTRN [115], TR-RL [116], CP-Bayes [117], TT-Bayes [118], TT-ADMM [119], BMF [120], [121]	6.2
		Hardware Speedup	TIE [122], LTNN [123], TT-Engine [124], Fast CP-CNN [125]	6.3
	Toolboxes	Basic Tensor Operations	Tensorly [126], TensorTools [127],Tensor Toolbox [128], TenDeC++ [129], OSTD [130], TensorD [131], TT-Toolbox [132], Tntorch [133], TorchMPS [134], ITensor [135],T3F [136], TensorNetwork [137], Scikit-TT [138]	7.1
		Deep Model Implementations	Tensorly-Torch [126], TedNet [64]	7.2
		Quantum Tensor Simulations	Yao [139], TensorNetwork [137], lambeq [140], ITensor [135], TeD-Q [141]	7.3

Neural Networks (NNs). NNs are biologically inspired learning paradigms that enable a machine to learn knowledge from observational data through backpropagation [38], [39]. NNs that are stacked with multiple layers, i.e., deep NNs (DNNs) [40], [41], are widely used in the field of artificial intelligence due to their powerful ability to capture abundant information from deep structures. Typical types of DNNs include restricted Boltzmann machines (RBMs) [42], convolutional NNs (CNNs) [41], [43], recurrent NNs (RNNs) [44], [45], and Transformers [46], [47]. DNNs currently reach state-of-the-art performance in a wide range of applications in computer vision [48] and natural language processing [49]. For example, a number of CNN architectures such as AlexNet [50], VGGNet [51], GoogLeNet [52] and ResNet [53] won championships on the ImageNet dataset [54], demonstrating good potential for solving image classification tasks. Particularly, Alphafold [55], [56], which is a kind of Transformer architecture, can identify the structure of a protein in days, which previously took researchers years. Recently, Alphafold2 [55], [56] predicted the structures of nearly all known proteins with average atomic precision. Deep learning techniques are still pushing forward advances in a number of disciplines, including speech recognition [57], DNA mutations detection [58], structural biology [55], [56], drug discovery [59], food security [60], etc.

Tensor Networks Meet Neural Networks. TNs and NNs are two types of networks that come from different origins and have achieved success from different aspects, as mentioned above. Interestingly, they are closely bonded through their multilinear mathematical property rather than being orthogonal to each other [11]. Therefore, a promising approach is to integrate them via multilinearity to attain the objective that “the whole is greater than the sum of the parts.” The main advantages of TNs are their compact structures, multiple entries, and close relationships with quantum mechanics, while NNs are well known for their wide applications [8], [12]. Based on these observations, it is feasible to combine TNs and NNs in three ways.

(1) Network Compression. NNs have achieved many successes in various tasks [40], [41], [41]. However, NNs still suffer from excess linear product calculations with massive numbers of dimensions and the curse of dimensionality [78]. A promising solution for addressing this issue is to utilize the lightweight and multilinear characteristics of TNs [68], [78], [79]. In detail, TNs can decompose any tensor of NNs into smaller blocks, thereby reducing the dimensionality to linear complexity [61], [62]. For example, in comparison to utilizing naive long short-term memory (LSTM) for action recognition tasks, TR-LSTM [79] models, which leverage TN technology to decompose weight tensors, can

compress the number of parameters by approximately 34,000 times while simultaneously outperforming naive LSTM.

(2) Information Fusion. In real-world data analysis cases, it is important to model higher-order interactions in data from multimodal sources to achieve better performance [8]. However, NNs are typically used to handle the inputs of one-mode vectors, so they lack sufficient expressive power to model such higher-order interactions [101]. To solve this problem, a promising approach is to embed TNs into NNs as efficient fusion units to process multimodal data with the help of the multiple-entry property [97], [98], [100]. Taking a visual question answering (VQA) task [142] as an example, multimodal Tucker fusion (MUTAN) [101] can learn high-level interactions between textual representations and visual representations via a Tucker-format framework. As a result, MUTAN has achieved state-of-the-art performance with an efficiently parameterized low-rank structure.

(3) Quantum Circuit Simulation. TNs can act as simulators and be bridges between classic NNs and quantum circuits. First, many studies have suggested implementing NNs on quantum circuits to accelerate their running speeds through the ultra-parallelism properties of quantum computation schemes [143], [144]. However, currently quantum computers are not sufficiently powerful for deploying NNs directly, which causes difficulty when verifying the possible performance of quantum neural networks (QNNs) [143]. Fortunately, TNs can be effective quantum simulators in electronic computers because of the equivalences between TNs and quantum circuits [8], [145]. In detail, the input qubits and unitary operation gates in quantum circuits can be viewed as tensors. Gate connections can also be viewed as tensor contractions in TN schemes [145]. By utilizing TNs to achieve quantum circuit simulation for NNs, a new era of QNNs exploration can be started before realistically powerful quantum computers are manufactured.

We call this family of approaches that connect TNs with NNs **tensorial neural networks (TNNs)**. To the best of our knowledge, this is the first comprehensive survey to bridge the connections among NNs, TNs, and Quantum Circuits. An overview of TNNs and their utility is shown in Table 1.

The remaining sections of this survey are organized as follows. Section 2 provides the fundamentals of tensor notations, tensor diagrams, and TN formats. Section 3 discusses the use of TNs for building compact TNNs. Section 4 explores efficient information fusion processes using TNNs. Section 5 discusses some basic applications of TNs in quantum circuits and TNNs. Section 6 explains some training and implementation techniques for TNNs. Section 7 introduces general and powerful toolboxes that can be used to process TNNs.

2 TENSOR BASIS

2.1 Tensor Notations

A tensor [146], [147], also known as a multiway array, can be viewed as a higher-order extension of a vector (i.e., a first-order tensor) or a matrix (i.e., a second-order tensor). Like the rows and columns in a matrix, an N th-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ has N modes (i.e., ways, orders, or indices) whose lengths (i.e., dimensions) are represented by I_1 to I_N , respectively. As shown in Table 2, lowercase letters denote scalars, e.g., a , boldface lowercase letters denote vectors, e.g., \mathbf{a} , boldface capital letters denote matrices, e.g., \mathbf{A} and boldface Euler script letters denote higher-order tensors, e.g., \mathcal{A} . In this paper, we define a “tensor” with a wider range that includes scalars, vectors, and matrices.

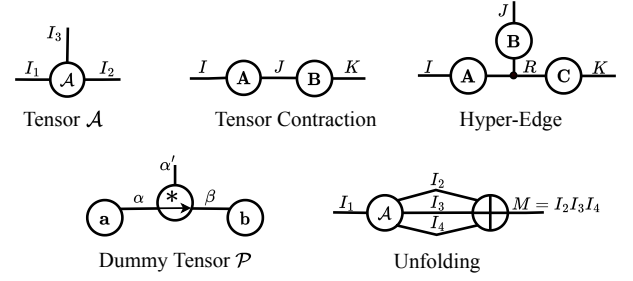


Fig. 1: Basic symbols for TN diagrams. For more basic knowledge about TNs, refer to [8] and [11].

TABLE 2: Tensor notations.

Symbol	Explanation
a	scalar
\mathbf{a}	vector
\mathbf{A}	matrix
\mathcal{A}	tensor
A	dimensionality
\otimes	convolution operation
\circ	outer product operation
$\langle \cdot, \cdot \rangle$	inner product of two tensors
$ \cdot\rangle$	quantum state bra vector(unit column complex vector)
$\langle \cdot $	quantum state ket vector(unit row complex vector)
$\langle \cdot \cdot \rangle$	inner product of two quantum state vectors

2.2 Tensor Diagrams

In this subsection, we introduce TN diagrams and their corresponding mathematical operations. TN diagrams were first developed by Roger Penrose [148] in the early 1970s and is now commonly used to describe quantum algorithms [8], [9] and machine learning algorithms [12], [61], [105]. In these diagrams, tensors are denoted graphically by nodes with edges [22]. TN diagrams are practical tools for the intuitive presentation and convenient representation of complex tensors. Therefore, tensor diagrams are widely used in the tensor field. As the data and weights in the deep learning field are all tensors, tensor diagrams are also promising for use as general network analysis tools in this area. An overview of the basic symbols of tensors is shown in Fig. 1.

2.2.1 Tensor Nodes

A tensor is denoted as a node with edges, as illustrated in Fig. 1. The number of edges denotes the modes of a tensor, and a value on an edge represents the dimension of the corresponding mode. For example, a one-edge node denotes a vector $\mathbf{a} \in \mathbb{R}^I$, a two-edge node denotes a matrix $\mathbf{A} \in \mathbb{R}^{I \times J}$ and a three-edge node denotes a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.

2.2.2 Tensor Contraction

Tensor contraction means that two tensors are contracted into one tensor along their associated pairs of indices. As a result, the corresponding connected edges disappear while the dangling edges persist. Tensor contraction can be formulated as a tensor product:

$$\mathcal{C} = \mathcal{A} \times_{M+1, M+2, \dots, M+N} \mathcal{B} \quad (1)$$

$$= \sum_{i_1, i_2, \dots, i_N} \mathcal{A}_{i_1, i_2, \dots, i_N, *} \mathcal{B}_{*, i_1, i_2, \dots, i_N}, \quad (2)$$

where $\mathcal{A} \in \mathbb{R}^{I_1 \times \dots \times I_N \times P_1 \times \dots \times P_K}$, $\mathcal{B} \in \mathbb{R}^{P_{K+1} \times \dots \times P_{K+M} \times I_1 \times \dots \times I_N}$, and $\mathcal{C} \in \mathbb{R}^{P_1 \times \dots \times P_K \times P_{K+1} \times \dots \times P_{K+M}}$. Fig. 1 also shows a diagram

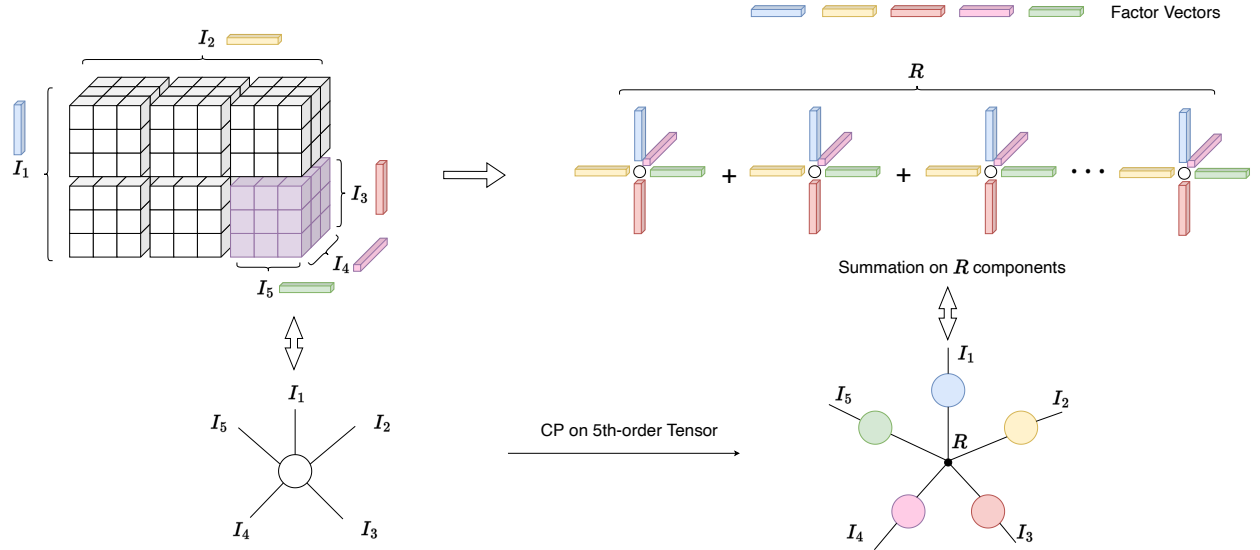


Fig. 2: Implementation of CP on a 5th-order tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4 \times I_5}$. We show the higher-order tensor in a matrix representation of size $I_1 \times I_2$, whose elements are 3rd-order tensors of size $I_2 \times I_3 \times I_4$. The main idea of CP is to decompose \mathcal{T} into five factor vectors that encode the modes of \mathcal{T} . Then, it is feasible to apply outer production on these factor vectors to recover \mathcal{T} . Moreover, by regarding such an outer production as a component, CP with a rank of R implements summation on R components to further improve the expressive power of the tensor.

of the matrix multiplication operation, which is the most classic tensor contraction situation. The equation representation is:

$$\mathbf{C} = \mathbf{A} \times_2^1 \mathbf{B}. \quad (3)$$

Tensor contractions among multiple tensors (e.g., TNs) can be computed by sequentially performing tensor contractions between each pair of tensors. It is worth mentioning that the contracting sequence must be determined to achieve better calculation efficiency [149].

2.2.3 Dummy Tensor

Recently, a newly designed dummy tensor was proposed by Hayashi et al. to represent convolution operations [61]. As depicted in Fig. 1, a node with star and arrow symbols denotes a dummy tensor. This operation is formulated as

$$\mathbf{y}_{j'} = \sum_{j=0}^{\alpha-1} \sum_{k=0}^{\beta-1} \mathcal{P}_{j,j',k} \mathbf{a}_j \mathbf{b}_k, \quad (4)$$

where $\mathbf{a} \in \mathbb{R}^\alpha$ denotes a vector that will be processed by a convolutional weight $\mathbf{b} \in \mathbb{R}^\beta$, and $\mathbf{y} \in \mathbb{R}^{\alpha'}$ is an output. $\mathcal{P} \in \{0, 1\}^{\alpha \times \alpha' \times \beta}$ is a binary tensor with elements defined as $\mathcal{P}_{j,j',k} = 1$ if $j = sj' + k - p$ and 0 otherwise, where s and p represent the stride and padding size, respectively. Thus, \mathcal{P} can be applied to any two tensors to form a convolutional relationship.

2.2.4 Hyperedge

As shown in Fig. 1, we illustrate the hyperedge that was also introduced by Hayashi et al. [61]. An example of a hyperedge with a size of R can be formulated as

$$\mathcal{Y}_{ijk} = \sum_{l=1}^R \mathbf{A}_{il} \mathbf{B}_{jl} \mathbf{C}_{kl}, \quad (5)$$

where $\mathbf{A} \in \mathbb{R}^{I \times R}$, $\mathbf{B} \in \mathbb{R}^{J \times R}$ and $\mathbf{C} \in \mathbb{R}^{K \times R}$ are three matrices. $\mathcal{Y} \in \mathbb{R}^{I \times J \times K}$ denotes the results of applying a hyperedge on \mathbf{A} , \mathbf{B} , and \mathbf{C} . A hyperedge node is simply equal to a tensor whose

diagonal elements are 1. This tensor indicates the addition operation performed over several substructures (e.g., the matrices in Fig. 1). Hayashi et al. [61] showed that a tensor diagram can represent an arbitrary tensorial CNN (TCNN) by introducing dummy tensors and hyperedges.

2.2.5 Tensor Unfolding

Tensor unfolding is an operation that virtually flattens a tensor into a high-dimensional but low-order tensor. Matricization is a special case of tensor unfolding. To be more specific, given an N th-order tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, its mode- n unfolding process yields a matrix $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times I_1 I_2 \dots I_{n-1} I_{n+1} \dots I_N}$. Such an operation can also be regarded as performing tensor contraction with a specifically designed tensor. A fourth-order tensor unfolding diagram is illustrated in Fig. 1.

2.3 Tensor Decomposition Formats

The commonly used terminology “tensor decomposition” (TD) is equivalent to “tensor network” to some extent. Previously, TD was employed primarily in signal processing fields [150], [151]. TNs were originally utilized largely in the physics and quantum circuit fields [8], [148]. Traditional TD models, such as CP [13], [14], [15] and Tucker decomposition [16], [17], can be viewed as basic kinds of TNs. In the realm of signal processing, several powerful TNs architectures for quantum analysis have also been introduced. For instance, MPS decomposition [152] was defined as TT decomposition [21] and has tremendous success in several applications [12]. After years of collaboration and progress across different research fields, there is no significant distinction between these two terminologies. Therefore, TD and TNs are treated in a unified way in this paper. We briefly introduce some basic TDs by employing TN diagrams.

2.3.1 CANDECOMP/PARAFAC

CP [13], [14], [15] factorizes a higher-order tensor into a sum of several rank-1 tensor components. For instance, given an N th-order

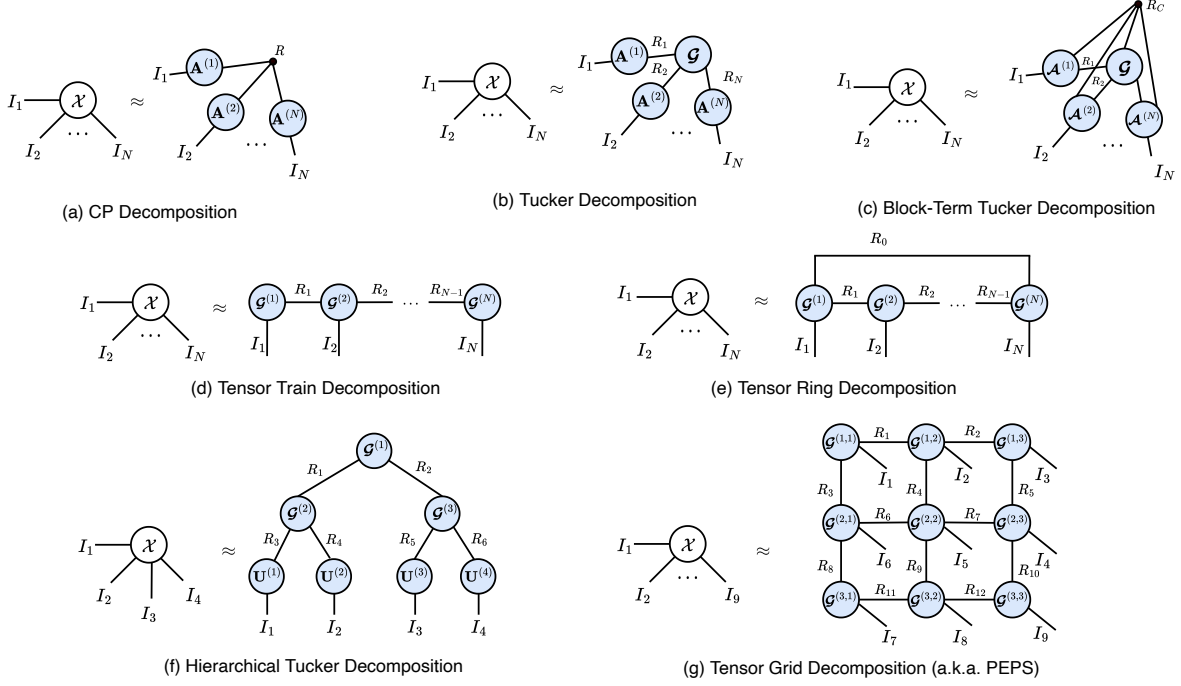


Fig. 3: TN diagrams of some popular decompositions. (a) Diagrams of the CP format. It decomposes a tensor \mathcal{X} into a sum of several rank-1 tensors $\mathbf{a}_{:,r}^{(1)} \circ \mathbf{a}_{:,r}^{(2)} \circ \dots \circ \mathbf{a}_{:,r}^{(N)}$. (b) Diagrams of Tucker decomposition. It decomposes a tensor \mathcal{X} into a core tensor \mathcal{G} multiplied by a matrix $\mathbf{A}^{(n)}$ along the n th mode. (c) Diagram of block term decomposition. It decomposes a tensor \mathcal{X} into a sum of several Tucker decompositions (on the right) with low Tucker ranks. (d) Diagram of TT decomposition. It decomposes a tensor \mathcal{X} into a linear multiplication of a set of 3rd-order core tensors $\mathcal{G}^{(2)} \dots \mathcal{G}^{(N-1)}$ and two matrices $\mathcal{G}^{(1)}, \mathcal{G}^{(N)}$. (e) Diagram of TR decomposition. It decomposes a tensor \mathcal{X} into a set of 3rd-order core tensors and contracts them into a ring structure. (f) Diagram of HT Decomposition. It represents a tensor \mathcal{X} as a tree-like diagram. For more basic knowledge about TNs, refer to [8] and [11].

tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, each of its elements in the CP format form can be formulated as

$$\mathcal{X}_{i_1, i_2, \dots, i_N} \approx \sum_{r=1}^R \mathcal{G}_r \prod_{n=1}^N \mathbf{A}_{i_n, r}^{(n)}, \quad (6)$$

where R denotes the CP rank (defined as the smallest possible number of rank-1 tensors [150]), \mathcal{G} denotes the diagonal core tensor (only the R nonzero elements on the superdiagonal) and $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R}$ denotes a series of factor matrices. The TN diagram for CP is illustrated in Fig. 3 (a). We also provide a detailed visualization of CP in Fig. 2 as an illustrative case of a TN.

When calculating a CP format, the first issue that arises is how to determine the number of rank-1 tensor components, i.e., the CP rank R . Actually, this is an NP-hard problem [153]. Hence, in practice, a numerical value is usually assumed in advance (i.e., as a hyperparameter), to fit various CP-based models [150]. After that, the diagonal core tensor \mathcal{G} and the factor matrices $\mathbf{A}^{(n)}$ can be directly solved by employing algorithmic iteration, which usually involves the alternating least-squares (ALS) method that was originally proposed in [13], [14].

2.3.2 Tucker Decomposition

Tucker decomposition [16], [17] factorizes a higher-order tensor into a core tensor multiplied by a corresponding factor matrix along each mode. To be more specific, given an N th-order tensor

$\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, the Tucker decomposition can be formulated in an elementwise manner as

$$\mathcal{X}_{i_1, i_2, \dots, i_N} \approx \sum_{r_1, \dots, r_N=1}^{R_1, \dots, R_N} \mathcal{G}_{r_1, r_2, \dots, r_N} \prod_{n=1}^N \mathbf{A}_{i_n, r_n}^{(n)}, \quad (7)$$

where $\{R_1, R_2, \dots, R_N\}$ denotes a series of Tucker ranks, $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_N}$ denotes the core tensor and $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ denotes a factor matrix. The TN diagram for Tucker decomposition is illustrated in Fig. 3 (b). Here, please note that compared with the CP rank, R_1, R_2, \dots, R_N can take different numerical values.

Tucker decomposition is commonly used and can be degraded to CP by setting the core tensor \mathcal{G} as a superdiagonal tensor whose diagonal elements are 1. In addition, Tucker decomposition lacks constraints on its factors, leading to the nonuniqueness of its decomposition results, which is typically undesirable for practical applications due to the lack of explainability. Consequently, orthogonal limitations are always imposed on the component matrices, yielding the well-known and classical higher-order singular value decomposition (HOSVD) algorithm [154].

2.3.3 BTT Decomposition

CP and Tucker decomposition both decompose a tensor into a core tensor multiplied by a matrix along each mode, while CP imposes an additional superdiagonal constraint on the core tensor for the sake of simplifying the structural information of the core tensor. A more generalized decomposition method called BTT decomposition [18] has been proposed to make a tradeoff between the CP and Tucker methods by imposing a block diagonal constraint

on Tucker's core tensor. The TN diagram for BTT decomposition is illustrated in Fig. 3 (c).

BTT decomposition aims to decompose a tensor into a sum of several Tucker decompositions with low Tucker ranks. Specifically, the BTT decomposition of a 4th-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times I_3 \times I_4}$ can be represented by 6 nodes with special contractions. Here, $\mathcal{G} \in \mathbb{R}^{R_C \times R_T \times R_T \times R_T \times R_T}$ denotes the R_C core tensors of the Tucker decompositions, and each $\mathcal{A}^{(n)} \in \mathbb{R}^{R_C \times I_n \times R_T}$ denotes the R_C corresponding factor matrices of the Tucker decompositions. Moreover, each element of \mathcal{X} is computed as

$$\mathcal{X}_{i_1, i_2, i_3, i_4} \approx \sum_{r_C=1}^{R_C} \sum_{r_1, r_2, r_3, r_4=1}^{R_T, R_T, R_T, R_T} \mathcal{G}_{r_C, r_1, r_2, r_3, r_4} \mathcal{A}_{r_C, i_1, r_1}^{(1)} \mathcal{A}_{r_C, i_2, r_2}^{(2)} \mathcal{A}_{r_C, i_3, r_3}^{(3)} \mathcal{A}_{r_C, i_4, r_4}^{(4)}, \quad (8)$$

where R_T denotes the Tucker rank (which means that the Tucker rank equals $\{R_T, R_T, R_T, R_T\}$) and R_C represents the CP rank. Together, they are called BT ranks.

The advantages of BTT decomposition mainly depend on its compatibility with the benefits of the both CP and Tucker methods. The reason for this is that when the Tucker rank is equal to 1, BTT decomposition degenerates to CP; when the CP rank equals 1, it degenerates to Tucker decomposition.

2.3.4 TT Decomposition

TT decomposition [21], [22], also called MPS decomposition in quantum physics [152], [155], is derived purely from TNs. TT decomposition factorizes a higher-order tensor into a linear multiplication of a series of 3rd-order core tensors. For example, given an N th-order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, the TT decomposition can be formulated in an elementwise manner as

$$\mathcal{X}_{i_1, i_2, \dots, i_N} \approx \sum_{r_1, r_2, \dots, r_{N-1}=1}^{R_1, R_2, \dots, R_{N-1}} \mathcal{G}_{1, i_1, r_1}^{(1)} \mathcal{G}_{r_1, i_2, r_2}^{(2)} \mathcal{G}_{r_2, i_3, r_3}^{(3)} \dots \mathcal{G}_{r_{N-1}, i_N, 1}^{(N)}, \quad (9)$$

where $\{R_1, R_2, \dots, R_{N-1}\}$ denote the TT ranks, $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ denotes a 3rd-order core tensor and $R_0 = R_N = 1$, which means that $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(N)}$ are actually two matrices. The TN diagram for TT decomposition is illustrated in Fig. 3 (d).

TT decomposition can be computed easily by employing SVD recursively. In addition, as the simplest model among the available TN, TT decomposition is widely applied in the theory and practice of TNs [9]. Notably, Eq. (9) and Fig. 3 (d) have an MPS format. Some papers [77], [89], [93] have also used TT decomposition with an MPO [156] format. Given a $2N$ -order tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times J_1 \times I_2 \times J_2 \times \dots \times I_N \times J_N}$, its MPO decomposition can be mathematically expressed as

$$\mathcal{X}_{i_1, j_1, i_2, j_2, \dots, i_N, j_N} \approx \sum_{r_1, r_2, \dots, r_{N-1}=1}^{R_1, R_2, \dots, R_{N-1}} \mathcal{G}_{1, i_1, j_1, r_1}^{(1)} \mathcal{G}_{r_1, i_2, j_2, r_2}^{(2)} \mathcal{G}_{r_2, i_3, j_3, r_3}^{(3)} \dots \mathcal{G}_{r_{N-1}, i_N, j_N, 1}^{(N)}, \quad (10)$$

where $\{R_1, R_2, \dots, R_{N-1}\}$ denote the ranks, $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ denotes a 4th-order core tensor and $R_0 = R_N = 1$, which means that $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(N)}$ are actually two 3rd-order core tensors.

2.3.5 TR Decomposition

TT benefits from fast convergence. However, it suffers from its two endpoints, which hinder the representation ability and flexibility of TT-based models. Thus, to release the power of a linear architecture, researchers link its endpoints to produce a ring format named a TR [25]. The TR decomposition of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ can be formulated as

$$\mathcal{X}_{i_1, i_2, \dots, i_N} \approx \sum_{r_0, r_1, \dots, r_{N-1}}^{R_0, R_1, \dots, R_{N-1}} \mathcal{G}_{r_0, i_1, r_1}^{(1)} \mathcal{G}_{r_1, i_2, r_2}^{(2)} \mathcal{G}_{r_2, i_3, r_3}^{(3)} \dots \mathcal{G}_{r_{N-1}, i_N, r_0}^{(N)}, \quad (11)$$

where $\{R_0, R_1, \dots, R_N\}$ denote the TR ranks, each node $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ is a 3rd-order tensor and $R_0 = R_N$. Compared with TT decomposition, it is not necessary for TR decomposition to follow a strict order when multiplying its nodes. The TN diagram for TR decomposition is illustrated in Fig. 3 (e).

2.3.6 HT Decomposition

HT decomposition [26] possesses a tree-like structure. In general, it is feasible to transfer a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times \dots \times I_N}$ to a binary tree with a root node associated with $S_{set} = \{1, I_2, \dots, N\}$ and $\mathcal{X} = \mathcal{U}_{S_{set}}$ as the root frame. $S_{set1}, S_{set2} \subseteq S_{set}$ is defined as the index set, which is associated with the left child node $\mathcal{U}_{S_{set1}}$ and right child node $\mathcal{U}_{S_{set2}}$. $\mathcal{U}_{S_{set1}} \in \mathbb{R}^{R_1 \times I_{\min(S_{set1})} \times \dots \times I_{\max(S_{set1})}}$ can also be recursively decomposed into its left child node $\mathcal{U}_{D_{set1}}$ and right child node $\mathcal{U}_{D_{set1}}$. The first three steps are as

$$\mathcal{U}_{S_{set}} \approx \mathcal{G}_s \times_1^2 \mathcal{U}_{S_{set1}} \times_1^2 \mathcal{U}_{S_{set2}}, \quad (12)$$

$$\mathcal{U}_{S_{set1}} \approx \mathcal{G}_{s1} \times_1^2 \mathcal{U}_{D_{set1}} \times_1^2 \mathcal{U}_{D_{set2}}, \quad (13)$$

$$\mathcal{U}_{S_{set2}} \approx \mathcal{G}_{s2} \times_1^2 \mathcal{U}_{D_{set3}} \times_1^2 \mathcal{U}_{D_{set4}}, \quad (14)$$

where $\mathcal{G}_s \in \mathbb{R}^{R_1 \times R_2}$, $\mathcal{G}_{s1} \in \mathbb{R}^{R_1 \times R_3 \times R_4}$ and $\mathcal{G}_{s2} \in \mathbb{R}^{R_2 \times R_5 \times R_6}$. This procedure can be performed recursively to obtain a tree-like structure. The TN diagram for HT decomposition is illustrated in Fig. 3 (f).

2.3.7 PEPS Decomposition

A TN structure with different typologies and higher-dimensional connections can also be considered. PEPS decomposition [8], [27], [28], also known as tensor grid decomposition [157], is a high-dimensional TN that generalizes a TT. PEPS decomposition provides a natural structure that can capture more high-dimensional information. PEPS cores can be characterized as $\mathcal{G}^{(m,n)} \in \mathbb{R}^{I_{mn} \times R_{l_{mn}} \times R_{r_{mn}} \times R_{u_{mn}} \times R_{d_{mn}}}$. The mathematical formula [75] is

$$\mathcal{X}_{i_1, i_2, \dots, i_{MN}} = \sum_{h^{(R)}, h^{(C)}} \sum_{m,n} \mathcal{G}_{i_{mn}; h_{l_{mn}}^{(R)}, h_{r_{mn}}^{(R)}, h_{u_{mn}}^{(C)}, h_{d_{mn}}^{(C)}}^{(m,n)}. \quad (15)$$

The indices are defined as

$$\begin{cases} l_{mn} = (n-2)M + m, \\ r_{mn} = (n-1)M + m, \\ u_{mn} = (m-2)N + n, \\ d_{mn} = (m-1)N + n, \\ R_i^{(R)} = 1, \quad \text{while } i < 0 \text{ or } i > M(N-1), \\ R_i^{(C)} = 1, \quad \text{while } i < 0 \text{ or } i > N(M-1), \end{cases} \quad (16)$$

where M and N are the numbers of rows and columns in the tensor cores, respectively, and $h_i^{(R)}$ and $h_j^{(C)}$ are ranks in the row direction and column direction, respectively. The TN diagram for PEPS

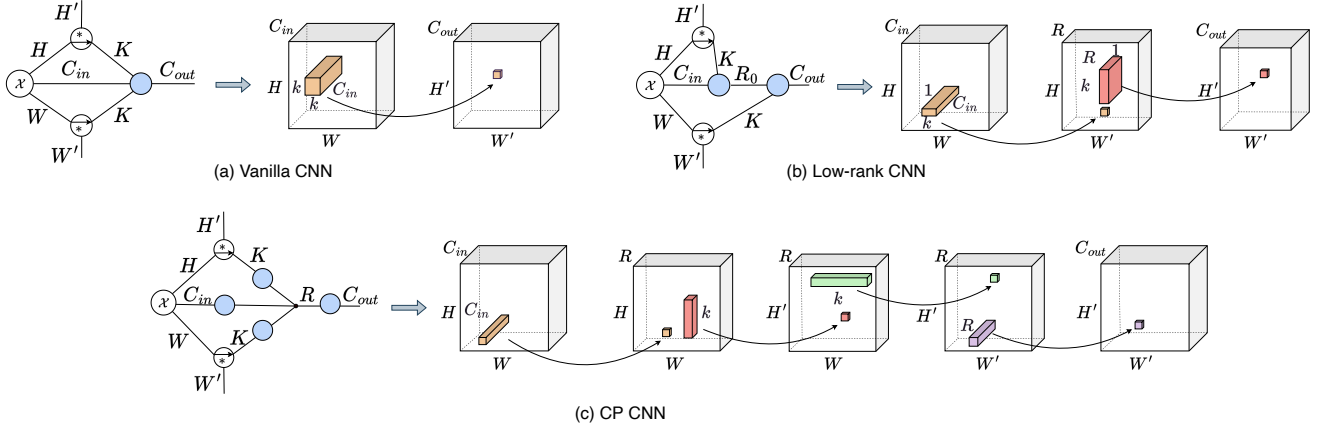


Fig. 4: Correspondence between TN diagrams and convolutional procedures. In each subfigure, the left part is a TN diagram, and the right part is the associated commonly used feature representation.

decomposition is illustrated in Fig. 3 (g). PEPS decomposition has a polynomial correlation decay with the separation distance. In contrast, MPS decomposition has an exponential correlation decay. This indicates that PEPS decomposition has a more powerful representation ability [8] because it strengthens the interactions between different tensor modes.

3 NETWORK COMPRESSION WITH TNNs

DNNs have extraordinarily high spatial and temporal complexity levels, as deeply stacked layers contain large-scale matrix multiplications. As a result, DNNs usually require several days for training while occupying a large amount of memory for inference purposes. In addition, large weight redundancy has been proven to exist in DNNs [158], indicating the possibility of compressing DNNs while maintaining performance. Motivated by this, a wide range of compression techniques have been developed, including pruning [159], [160], quantization [161], [162], distillation [163], [164] and low-rank decomposition [79], [165], [166]. Among them, applying TNNs to DNNs to construct TNNs can be a good choice since TNNs have excellent abilities to approximate the original weights with many fewer parameters [113]. In this direction, researchers have completed many studies, especially concerning the reconstruction of convolutional and fully connected layers through a variety of TD formats [61], [67], [69], [79]. With compact architectures, these TNNs can achieve improved performance with less redundancy. In this section, we introduce five common kinds of TNNs, i.e., TCNNs in Section 3.1, tensorial RNNs (TRNNs) in Section 3.2, tensorial Transformers in Section 3.3, tensorial GNN (TGNN) in Section 3.4, and tensorial RBMs in Section 3.5.

3.1 TCNNs

CNNs have recently achieved much success. However, CNNs' enormous sizes cause weight redundancy and superfluous computations, affecting both their performance and efficiency. TD methods can be effective solutions to this problem. Commonly, CNNs represented with tensor formats are called TCNNs. Prior to introducing TCNNs, we formulate a vanilla CNN, shown in Fig. 4 (a), as

$$\mathbf{Y} = \mathbf{X} \circledast \mathbf{C} + \mathbf{b}, \quad (17)$$

where $\mathbf{C} \in \mathbb{R}^{K \times K \times I \times O}$ denotes a convolutional weight, $\mathbf{X} \in \mathbb{R}^{I \times H \times W}$ denotes an input, $\mathbf{Y} \in \mathbb{R}^{O \times H' \times W'}$ denotes an output,

$\mathbf{b} \in \mathbb{R}^O$ represents a bias, and \circledast denotes a convolutional operator. K represents the kernel window size, I is an input channel, H and W denote the height and width of \mathbf{X} , O is an output channel, and H' and W' denote the height and width of \mathbf{Y} , respectively. TCNNs mainly focus on decomposing channels I and O . In detail, the weight \mathbf{C} is first reshaped to $\tilde{\mathbf{C}} \in \mathbb{R}^{K \times K \times I_1 \times I_2 \times \dots \times I_M \times O_1 \times O_2 \times \dots \times O_N}$, where $\prod_{k=1}^M I_k = I$ and $\prod_{k=1}^N J_k = J$. Then, TCNNs can be derived by tensorizing the reshaped convolutional kernel $\tilde{\mathbf{C}}$.

To accelerate the CNN training and inference process, CP-CNN [61], [62], [63] is constructed by decomposing the convolutional weight into the CP format, as shown in Fig. 4 (d). CP-CNN only contains vectors as subcomponents, leading to an extremely compact structure and the highest compression ratio. As with CP-CNN, it is possible to implement additional TCNNs by applying tensor formats (as seen in the examples in Fig. 3) to the convolutional weight. Tucker decomposition, a widely used tensor format, is often applied to CNNs to form Tucker-CNNs [64], [65]. Different from simple Tucker formats, a BTT-CNN has a hyperedge R_c , which can denote the summation of Tucker decompositions. Other BTT-CNNs [69] have also been proposed. Compared to Tucker CNNs, BTT-CNNs are much more powerful and usually derive better results [69]. Highly compact TT formats have also been introduced to CNNs to implement TT-CNNs [66]. Compared to TTs, TR formats are usually much more compact [68], and TR-CNNs [68] are much more powerful than TT-CNNs.

There are also some tensorial convolutional neural networks that decompose more than just the convolution cores. The tensorized network (T-Net) [73] treats the whole network as a one-layer architecture and then decomposes it. As a result, the T-Net achieves better results with a lighter structure. CP-higher-order convolution (CP-HOConv) [76] utilizes the CP format to handle tasks with higher-order data, e.g., spatiotemporal emotion estimation. For multitask missions, Yang et al. [74] proposed the Tensor Train multitask (TTMT) and Tucker multitask (TMT) models using TT and Tucker formats, respectively, to alleviate the negative transfer problem in a hard sharing architecture and reduce the parameter volume in a soft structure. A PEPS-like concatenated TN layer [75] for multitask missions was also proposed. Unlike the TTMT and TMT models, which suffer from the negative transfer problem due to their hard sharing architectures, the PEPS structure only contains a soft sharing layer, thereby achieving better performance.

3.2 TRNNs

RNNs, such as the vanilla RNN and LSTM, have achieved promising performance on sequential data. However, when dealing with high-dimensional input data (e.g., video and text data), the input-to-hidden and hidden-to-hidden transformations in RNNs will result in high memory usage rates and computational costs. To solve this problem, low-rank TD is efficient for compressing the transformation process in practice. First, we formulate an RNN as

$$\mathbf{h}^{(t+1)} = \phi(\mathbf{W}\mathbf{x}^{(t)} + \mathbf{U}\mathbf{h}^{(t)} + \mathbf{b}), \quad (18)$$

where $\mathbf{h}^{(t)} \in \mathbb{R}^O$ and $\mathbf{x}^{(t)} \in \mathbb{R}^I$ denote the hidden state and input feature at time t , respectively, $\mathbf{W} \in \mathbb{R}^{O \times I}$ is the input-to-hidden matrix, $\mathbf{U} \in \mathbb{R}^{O \times O}$ represents the hidden-to-hidden matrix, and $\mathbf{b} \in \mathbb{R}^O$ is a bias. $\phi(\cdot)$ indicates a series of operations that form RNN variants, including the vanilla RNN and LSTM [167]. Eq. (18) can also be reformulated in a concatenation form that is widely used in TD:

$$\mathbf{h}^{(t+1)} = \phi([\mathbf{W}, \mathbf{U}][\mathbf{x}^{(t)}, \mathbf{h}^{(t)}] + \mathbf{b}), \quad (19)$$

where $[\mathbf{W}, \mathbf{U}] \in \mathbb{R}^{O \times (I+O)}$ and $[\mathbf{x}^{(t)}, \mathbf{h}^{(t)}] \in \mathbb{R}^{(I+O)}$ denote the concatenation of \mathbf{W}, \mathbf{U} and $\mathbf{x}^{(t)}, \mathbf{h}^{(t)}$, respectively. As shown in Fig. 5, there are usually two ways to decompose RNNs: (a) only tensorizing \mathbf{W} , which is often the largest component in an RNN, and (b) tensorizing $[\mathbf{W}, \mathbf{U}]$ for extreme compression. Note that since \mathbf{U} is usually smaller than \mathbf{W} , no works decompose \mathbf{U} only. The process of implementing a TRNN is the same as that used to implement a TCNN, namely, reshaping the weights into higher-order formulations and replacing them with tensor formats.

The most direct and simple compression method is to solely decompose the enormous input-to-hidden matrix \mathbf{W} . The CP-RNN and Tucker-RNN [64] can be directly constructed with the CP and Tucker formats, respectively. With an extremely compact low-rank structure, the CP-RNN can always derive the smallest size in comparison with other tensor formats. The TT-RNN [77] implements the TT format on an RNN to obtain a high parameter compression ratio. However, the TT-RNN suffers from a linear structure with two smaller endpoints, which hinders the representation ability and flexibility of TT-based models. To release the power of a linear architecture, TRs were proposed to link the endpoints to create a ring format [25]. TR-An RNN [79] with a TR was formed to achieve a much more compact network. BTT-RNN [69], [78] was constructed on the generalized TD approach: BTT decomposition [19]. BTT-RNN can automatically learn interparameter correlations to implicitly prune redundant dense connections and simultaneously achieve better performance.

Moreover, studies are utilizing TD to compress an RNN's two transformation layers, and some have even developed decomposition methods that are suitable for both RNNs and CNNs. TT-GRU [82] and the HT-RNN [80] decompose $[\mathbf{W}, \mathbf{U}]$ to attain a higher compression ratio. Specifically, TT-GRU [82] applies a TT for decomposition, and the HT-RNN [80] adopts HT decomposition. Unlike prior works that decomposed hidden matrices, Conv-TT-LSTM [84] utilizes the idea of a TT to represent convolutional operations. As shown in Fig. 5, through a TT-like convolution, Conv-TT-LSTM can replace convolutional LSTM with fewer parameters while achieving good results on action benchmarks. For the adaptation of both CNNs and RNNs, a hybrid TD (termed HT-TT) method that combines HT and TT decomposition [72] was adopted to compress both the CNN and RNN $[\mathbf{W}, \mathbf{U}]$ matrices. In addition, the tensor contraction layer (TC-Layer) [71] was designed

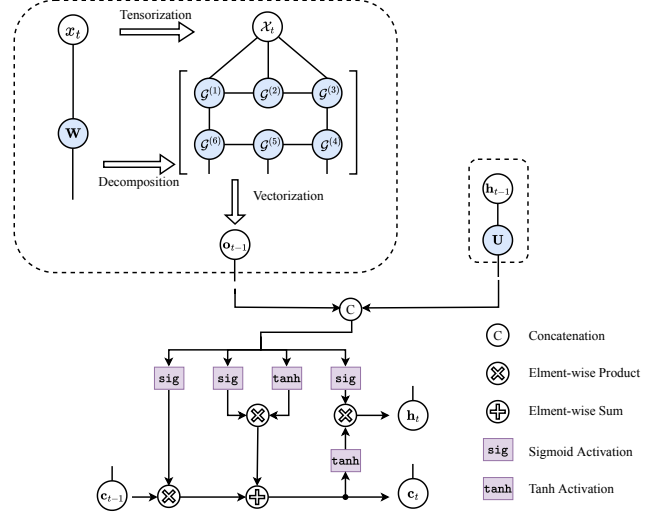


Fig. 5: TR LSTM. It is effective at reducing the parameters of an LSTM model by replacing the input-to-hidden transformation weights with TR decomposition.

to replace the fully connected layer and therefore can be utilized as the last layer of a CNN and the hidden layers in RNNs. Interestingly, TC-Layer is a special case of a TT-based layer obtained by setting the ranks to 1.

3.3 Tensorial Transformers

Transformers [46], [168] are well known for processing sequence data. Compared with CNNs and RNNs, Transformers can be stacked into large-scale sizes to achieve significant performance [47]. However, Transformers are still redundant, similar to classic DNNs, which can be made smaller and more efficient [88]. Therefore, TD, as a flexible compression tool, can be explored to reduce the numbers of parameters in Transformers [85], [86], [87].

Classic Transformers mainly consist of self-attention (SA) and feedforward Networks (FFNs). SA processes the given query matrix \mathbf{Q} , key matrix \mathbf{K} and value matrix \mathbf{V} with parameters $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O$. More generally, SA is separated into n heads: $\{\mathbf{W}_i^Q\}^n, \{\mathbf{W}_i^K\}^n, \{\mathbf{W}_i^V\}^n, \{\mathbf{W}_i^O\}^n$. Each head can be calculated as

$$\text{Att}_i(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{W}_i^Q \mathbf{W}_i^K^T \mathbf{K}^T}{\sqrt{d}} \right) \mathbf{V}\mathbf{W}_i^V \mathbf{W}_i^O^T. \quad (20)$$

Then, $\text{SA}((\mathbf{Q}, \mathbf{K}, \mathbf{V})) = \sum_{i=1}^n \text{Att}_i(\mathbf{Q}, \mathbf{K}, \mathbf{V})$. Another important component, the FFN, is formulated as

$$\text{FFN}(\mathbf{X}) = \text{ReLU}(\mathbf{X}\mathbf{W}^{in} + \mathbf{b}^{in})\mathbf{W}^{out} + \mathbf{b}^{out}, \quad (21)$$

where \mathbf{X} is the input, \mathbf{b}^{in} and \mathbf{b}^{out} are biases, and \mathbf{W}^{in} and \mathbf{W}^{out} are weights. Apparently, the number of parameters in a Transformer is mainly based on its linear transformation matrices, i.e., $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V, \mathbf{W}^O, \mathbf{W}^{in}$ and \mathbf{W}^{out} .

Therefore, most compression studies focus on eliminating the parameters of these matrices. For instance, the MPO structure was proposed to decompose each matrix in a Transformer [86], generating central tensors (containing the core information) and small auxiliary tensors. A tuning strategy was further adopted to continue training the auxiliary tensors to achieve a performance

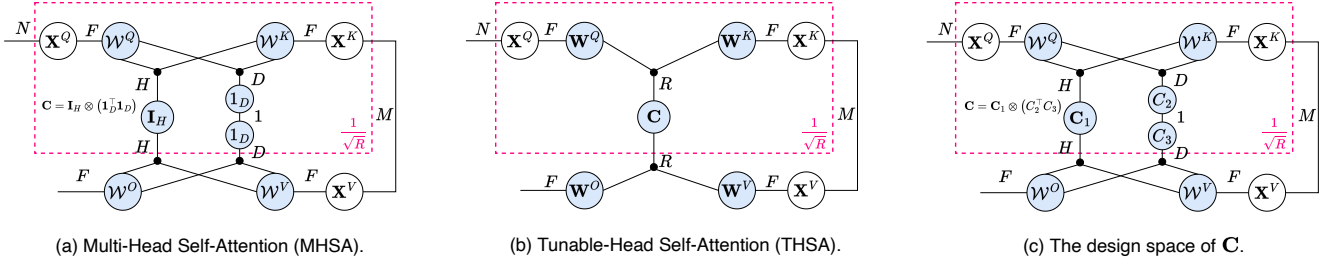


Fig. 6: Tensor diagrams for SA modules [86]. (a) It is feasible to represent a classic multihead SA (MHSA) mechanism in a tensor diagram. MHSA can be treated as a special case of tunable-head self-attention (THSA) by setting $\mathbf{C} = \mathbf{I}_H \otimes (\mathbf{1}_D^T \mathbf{1}_D)$. (b) The THSA of the Tuformer can be a more generalized version of SA through a trainable matrix \mathbf{C} . (c) THSA has a design space formulated as $\mathbf{C} = \mathbf{C}_1 \otimes (\mathbf{C}_2^T \mathbf{C}_3)$, which is the direct generalized form of MHSA.

improvement while freezing the weight of the central tensor to retain the main information of the original matrix. Moreover, observing that a low-rank MPO structure can cause a severe performance drop, Hypoformer [89] was proposed based on hybrid TT decomposition; this approach concatenates a dense matrix part with a low-rank MPO part. Hypoformer retains the full-rank property while reducing the required numbers of operations and parameters to compress and accelerate the base Transformer. In addition, by concatenating all matrices into one larger tensor, Tucker-Bert [88] decomposes the concatenated tensor with Tucker decomposition to greatly reduce the number of parameters, leading to extreme compression and maintaining comparably good results. Interestingly, Tuformer [87] generalizes MHSA into the Tucker form, thus containing more expressive power and achieving better results, as shown in Fig. 6.

3.4 TGNNs

GNNs have achieved groundbreaking performances across a range of applications and domains [169]. One classic GNN layer consists of an aggregation function for aggregating the neighbor node information and an update function for updating the current node information. For example, the processing step for node v in the k -th layer of a GNN can be formulated as

$$\begin{aligned} \mathbf{a}_v^{(k)} &\leftarrow \text{Aggregate}_{(k)} \left(\left\{ \mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v) \right\} \right), \\ \mathbf{h}_v^{(k)} &\leftarrow \text{Update}_{(k)} \left(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)} \right), \end{aligned} \quad (22)$$

where $\mathbf{a}_v^{(k)}$ is an aggregated embedding vector, $\mathbf{h}_v^{(k-1)}$ is a node embedding vector, and $\mathcal{N}(v)$ is a neighbor node set. A typical choice for the update function is a simple-layer perceptron, and simple summation/maximization is always chosen as the aggregation function. Classic GNNs suffer from low model expressivity since high-order nonlinear information among nodes is missed [90]. Because of the merits of the tradeoff between expressivity and computing efficiency, the usage of TGNNs for graph data processing is quite beneficial.

To efficiently parameterize permutation-invariant multilinear maps for modeling the interactions among neighbors in an undirected graph structure, a TGNN [90] makes use of a symmetric CP layer as its node aggregation function. It has been demonstrated that a TGNN has a strong capacity to represent any multilinear polynomial that is permutation-invariant, including the sum and mean pooling functions. Compared to undirected graph processing, TGNNs are more naturally suited for high-order graph structures, such as knowledge graphs. Traditional relational graph

convolutional networks neglect the trilinear interaction relations in knowledge graphs and additively combine the information possessed by entities. The TGCN [91] was proposed by using a low-rank Tucker layer as the aggregation function to improve the efficiency and computational space requirement of multilinear modeling. TGNNs are also appropriate for high-order correlation modeling in dynamic spatial-temporal graph processing situations. For example, The DSTGNN [92] applies learnable TTG and STG modules to find dynamic time relations and spatial relations, respectively. Then, the DSTGNN explores the dynamic entangled correlations between the STG and TTG modules via a PEPS layer, which reduces the number of DSTGNN parameters.

3.5 Tensorial RBMs

RBMs [42] are generative stochastic NNs that can learn a probability distribution from an input set. A standard RBM consists of one visible factor $\mathbf{v} \in \mathbb{R}^M$ and one hidden factor $\mathbf{h} \in \mathbb{R}^N$ and assigns the following energy function for a joint vector $\{\mathbf{v}, \mathbf{u}\}$ as

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T \mathbf{W} \mathbf{h} - \mathbf{v}^T \mathbf{b} - \mathbf{c}^T \mathbf{h}, \quad (23)$$

where $\mathbf{b} \in \mathbb{R}^M$ and $\mathbf{c} \in \mathbb{R}^N$ are the biases of the visible layer and hidden layer, respectively, and $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the mapping weight matrix. The probability distribution of the joint vector $\{\mathbf{v}, \mathbf{u}\}$ can be defined as

$$P(v, h) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})}, \quad (24)$$

where Z is a partition function. Then, some loss function can be defined via the learnable distribution formulation to optimize the parameters. The RBM parameters appear to be based mostly on the mapping weight matrix.

As a result, the majority of compression research concentrates on reducing the number of weight matrix variables. For instance, Tv-RBM [95] explores the use of an RBM for higher-order inputs. In this model, the weight matrix is transformed into a CP layer structure, where each visible layer is represented as a tensor while each hidden layer is still a vector. In another higher-order RBM, namely, Mv-RBM [96], its visible and hidden layers are all represented as matrices, and its weights are represented as a TC layer [71]. MPO-RBM [170] and TT-RBM [93] represent the weight matrix with a TT layer to greatly compress the number of required parameters. Moreover, TR-RBM [94] performs TR decomposition on its RBM, where the visible and hidden layers of TR-RBM are all generalized to tensors.

Remark. Compact TNNs have demonstrated the potential to achieve extremely high compression ratios while preserving their

model performance. However, their computational acceleration rates are not very significant compared with their compression ratios, which is mainly due to the contraction operations. This therefore calls for further research to improve the employed contraction strategies, since unoptimized contraction strategies can result in unsatisfactory running memory consumption.

4 INFORMATION FUSION VIA TNNs

In real-world data analyses, the collected data can be derived from multiple sources; e.g., vision, sound, and text sources can be contained in video data [142]. For example, in the VQA task, the key point lies in effectively modeling the interactions between the two modalities, i.e., text and image information. When processing such data, it is infeasible to consider diverse sources in the same form. Therefore, it is desirable to mix this information through multiple entrances to address multimodal sources in special building structures. Such methods with entrances are called information fusion approaches. Feature-level fusion [171] and decision-level fusion [172] are popular methods that are used in the early stage. However, these methods are simple linear methods and do not allow intramodality dynamics to be efficiently modeled. To solve this problem, TNNs are utilized in fusion tasks for modeling intramodality dynamics based on the natural multilinear property. In addition, TNNs are capable of processing higher-order data, which is a widely used ability. In conclusion, TNs provide effective frameworks for tensor operations, and it is natural and meaningful to express and generalize the information fusion modules (such as attention modules and vector concatenation modules) encountered in deep learning through TNs. Therefore, many studies adopt TNNs to capture the higher-order interactions among data or parameters. In this section, we introduce two main series of TNN structures for information fusion: the tensor fusion layer in Section 4.1 and multimodal pooling in Section 4.2.

4.1 Tensor Fusion Layer-Based Methods

Multimodal sentiment analysis is a task containing three communicative modalities, i.e., the textual modality, visual modality, and acoustic modality [97]. Addressing multimodal sentiment analysis, Zadeh et al. [97] proposed novel TNNs with deep information fusion layers named tensor fusion layers (TFLs), which can easily learn intramodality dynamics and intermodality dynamics and are able to aggregate multimodal interactions, thereby efficiently fusing the three communicative modalities. Specifically, a TFL first takes embedded feature vectors \mathbf{z}_t , \mathbf{z}_v and \mathbf{z}_a derived by embedding networks rather than the original three data types. Then, the TFL concatenates a scalar 1 with each embedded feature vector:

$$\mathbf{z}'_t = \begin{bmatrix} \mathbf{z}_t \\ 1 \end{bmatrix}, \mathbf{z}'_v = \begin{bmatrix} \mathbf{z}_v \\ 1 \end{bmatrix}, \mathbf{z}'_a = \begin{bmatrix} \mathbf{z}_a \\ 1 \end{bmatrix}. \quad (25)$$

Then, as shown in Fig. 7, the TFL obtains a feature tensor \mathcal{Z} by calculating the outer product among the three concatenated vectors:

$$\mathcal{Z} = \mathbf{z}'_t \circ \mathbf{z}'_v \circ \mathbf{z}'_a = \begin{bmatrix} \mathbf{z}_t \\ 1 \end{bmatrix} \circ \begin{bmatrix} \mathbf{z}_v \\ 1 \end{bmatrix} \circ \begin{bmatrix} \mathbf{z}_a \\ 1 \end{bmatrix}. \quad (26)$$

Finally, the TFL processes the feature tensor \mathcal{Z} to obtain a prediction \mathbf{y} via a two-layer fully connected NN. Compared to direct concatenation-based fusion, which only considers unimodal interactions [97], the TFL benefits from capturing both unimodal interactions and multimodal interactions.

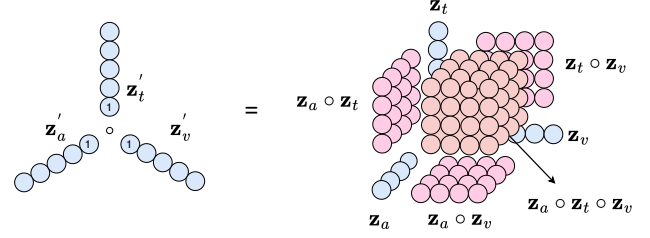


Fig. 7: Illustration of the tensor fusion process in Eq. (26). Different from a TN diagram, each circle corresponds to a value.

Despite its success, the TFL suffers from exponential increases in its computational complexity and number of parameters when the number of modalities increases. For example, in a multimodal sentiment analysis case [97], the feature tensor $\mathcal{Z} \in \mathbb{R}^{129 \times 33 \times 33}$ and the hidden vector $\mathbf{h} \in \mathbb{R}^{128}$ can result in 17,981,568 parameters to be optimized. To address these excessive parameters, low-rank multimodal fusion (LMF) [99] adopts a special BTT layer to overcome the massive computational cost and overfitting risks of the TFL. For a general situation with n modalities, the feature tensor $\mathcal{Z} = \circ_{m=1}^M \mathbf{z}_m$ can be processed. The hidden vector \mathbf{h} can be computed as follows:

$$\mathbf{h} = \text{ReLU} \left(\mathbf{h}e(\mathbf{W}_1 \mathbf{z}'_1, \mathbf{W}_2 \mathbf{z}'_2, \dots, \mathbf{W}_M \mathbf{z}'_M, I) + \mathbf{b} \right),$$

where $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_h}$ is the weight matrix and $I \in \mathbb{R}^{d_h \times d_h}$ is an identity matrix. LMF reduces the computational complexity of the TFL from $O\left(\prod_{m=1}^M d_m\right)$ to $O\left(d_h \times \sum_{m=1}^M d_m\right)$.

Although LMF and the TFL achieve better fusion results than other methods, they restrict the order of interactions, causing higher-order interactions to lack information. A PTP [100] block has been proposed to tackle this problem. The whole procedure and TN diagram of PTP are shown in Fig. 8 and Fig. 9, respectively.

PTP first merges all feature vectors $\{\mathbf{z}_m\}_{m=1}^M$ into a long feature vector

$$\mathbf{z}_{12\dots M}^\top = [1, \mathbf{z}_1^\top, \mathbf{z}_2^\top, \dots, \mathbf{z}_M^\top]. \quad (27)$$

The polynomial feature tensor of degree P is represented as

$$\mathcal{Z}^P = \mathbf{z}_{12\dots M} \circ \mathbf{z}_{12\dots M} \circ \dots \circ \mathbf{z}_{12\dots M}. \quad (28)$$

PTP [100] then adopts a tensorial layer (e.g., a CP layer) to process the polynomial feature tensor \mathcal{Z}^P . The CP layer is represented as

$$\mathbf{h} = \mathbf{h}e(\mathbf{W}_1 \mathbf{z}_{12\dots M}, \dots, \mathbf{W}_P \mathbf{z}_{12\dots M}, \mathbf{\Lambda}) \quad (29)$$

where $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_h}$ is the weight matrix and $\mathbf{\Lambda} \in \mathbb{R}^{d_h \times d_h}$ is a learnable diagonal matrix. The structure of PTP is also equivalent to that of a deep polynomial NN [173]. PTP models all nonlinear high-order interactions. For multimodal time series data, one approach uses a “window” to characterize local correlations and stack the PTP blocks in multiple layers. Such a model is called a hierarchical polynomial fusion network (HPFN) [100]. The HPFN can recursively process local temporal-modality patterns to achieve a better information fusion effect.

The structure of a single-layer PTP block is similar to that of a shallow convolutional arithmetic circuit (ConvAC) network [107] (see Section 5.3). The only difference between ConvAC and PTP is that the standard ConvAC network processes quantum location features, whereas PTP processes the temporal-modality patterns and polynomial concatenated multimodal features. The

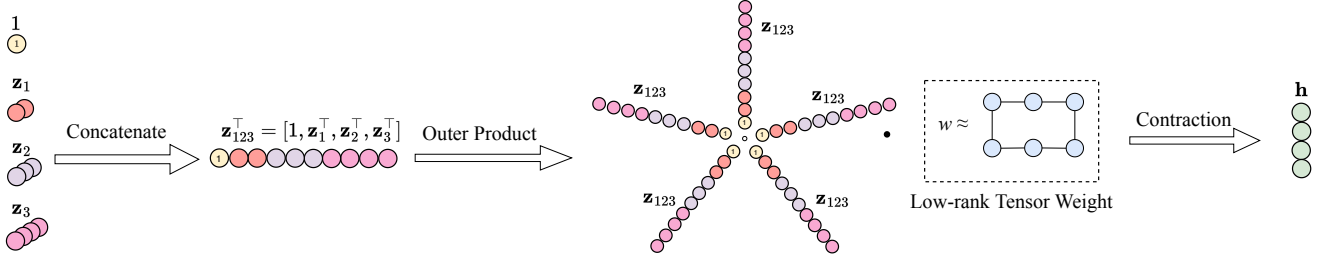


Fig. 8: Illustration of polynomial tensor pooling (PTP) [100]. PTP first concatenates all feature vectors $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3$ into a longer feature vector $\mathbf{z}_{123}^\top = [1, \mathbf{z}_1^\top, \mathbf{z}_2^\top, \mathbf{z}_3^\top]$, then derives a polynomial feature tensor by repeatedly performing outer product operations on the feature vector \mathbf{z}_{123} and finally adopts a tensorial layer (e.g., a TR layer) to merge the polynomial feature tensor into a vector \mathbf{h} .

HPFN is nearly equivalent to a deeper ConvAC network, and its great expressive power might be implied by their connection. The recursive relationships in deep polynomial NNs have also been found and implemented so that polynomial inputs can be efficiently computed via a hierarchical NN [100]. Chrysos et al. [173] also discovered similar results.

4.2 Multimodal Pooling-Based Methods

Another group of information fusion methods originated from VQA tasks [142]. In VQA tasks, the most important aspect is to parameterize bilinear the interactions between visual and textual representations. To address this aspect, some tensor fusion methods have been discovered in this area. Multimodal compact bilinear pooling (MCB) [102] is a well-known fusion method for VQA tasks and can be regarded as a special Tucker decomposition-based NN. MCB tries to optimize the simple bilinear fusion operation

$$\mathbf{z} = \mathbf{W}[\mathbf{v} \circ \mathbf{q}], \quad (30)$$

where \mathbf{v} and \mathbf{q} are input vectors with different modalities and \mathbf{W} is a learnable weight matrix. Moreover, MCB optimizes the computational cost of the outer product operation based on the property of the count sketch projection function.

Multimodal low-rank bilinear pooling (MLB) [103] adopts a CP layer in a data fusion step that can be formulated as follows:

$$\mathbf{z} = \mathbf{1}^T (\mathbf{W}_v \mathbf{v} \circ \mathbf{W}_q \mathbf{q}), \quad (31)$$

where \mathbf{W}_q and \mathbf{W}_v are preprocessing weight matrices for inputs \mathbf{q} and \mathbf{v} , respectively and $\mathbf{1}$ is a vector in which all values are 1. The structure of the MLB method is a special case of LMF (see Sec. 4.1). MLB fusion methods can also be regarded as simple product pooling when the number of modalities is equal to two.

MUTAN [101] is a generalization of MCB and MLB. MUTAN adopts a Tucker layer to learn the bilinear interactions between visual and textual features:

$$\begin{aligned} \mathbf{z} &= \left((\mathcal{J}_c \times_1^1 (\mathbf{q}^\top \mathbf{W}_q)) \times_2^1 (\mathbf{v}^\top \mathbf{W}_v) \right) \times_3^1 \mathbf{W}_o, \\ \mathbf{z} &= (\mathcal{J}_c \times_1^1 \tilde{\mathbf{q}}) \times_2^1 \tilde{\mathbf{v}}, \end{aligned} \quad (32)$$

where $\tilde{\mathbf{q}} = \tanh(\mathbf{q}^\top \mathbf{W}_q)$ and $\tilde{\mathbf{v}} = \tanh(\mathbf{v}^\top \mathbf{W}_v)$, \mathcal{J}_c is the fusion weight tensor, and \mathbf{W}_o is the output processing weight matrix. Moreover, MUTAN [101] adopts a low rank for the fusion weight tensor \mathcal{J}_c , as follows:

$$\mathcal{J}_c[:, :, k] = \sum_{r=1}^R \mathbf{m}_r^k \circ \mathbf{n}_r^{k\top}, \quad (33)$$

where \mathbf{m}_r^k and $\mathbf{n}_r^{k\top}$ are weight vectors and R is the number of ranks. MUTAN can represent comprehensive bilinear interactions while maintaining a reasonable model size by factorizing the interaction tensors into interpretable elements.

Furthermore, compact trilinear interaction (CTI) [104] was proposed to use an attention-like structure. Instead of presenting the given data as a single vector, this method represents every modality as a matrix $A \in \mathbb{R}^{n_1 \times d_a}$, where d_a corresponds to the feature dimension and n_1 denotes the number of states. CTI simultaneously learns high-level trilinear joint representations in VQA tasks and overcomes both the computational complexity and memory issues in trilinear interaction learning [104].

Remark. When fusing information in many multimodal tasks, TNNs can achieve promising results with natural multilinear and compact frameworks. However, the high-order outer product operation used in TNNs may cause unexpected computational complexity increases and even unstable numerical properties. Therefore, it is important to consider an efficient algorithm for reducing memory consumption and apply a feasible initialization algorithm (e.g., [114]) to achieve good stability.

5 QUANTUM CIRCUIT SIMULATION WITH TNNs

In the past few years, the development of quantum computing theory has attracted much attention [174], [175], [176]. Quantum systems have superiority in terms of parallelism over classic electronic computers [177], so they can achieve algorithms with lower time complexity. For example, Shor's algorithm [178] based on quantum systems is theoretically exponentially faster than the classic prime number decomposition algorithm. Quantum circuits are computational hardware implements of quantum systems, and they theoretically correspond to TNs and TNNs [179], [180]. Quantum states are mathematical entities of quantum systems and are consistent with higher-order tensors with some constraints [8]. Therefore, TNNs can be used as simulators in classic computers to model realistic quantum circuits [8], [145]. Taking advantage of the ultrahigh parallelism of quantum computing, some special TNNs can be implemented on small, near-term quantum devices [180]. Quantum circuit simulation on TNNs mainly focuses on the roles of TNs as bridges between classic NNs and QNNs rather than the more general TN-based quantum circuit simulation paradigm. Please refer to other papers [8], [30], [145] if readers are interested in general circuit simulation via TNs. In this section, we use the term "classic data" to denote the data in classic electronic computers. We introduce methods for mapping classic data to quantum states through TNs in Section 5.1, then introduce basic supervised and unsupervised processing methods for the mapped

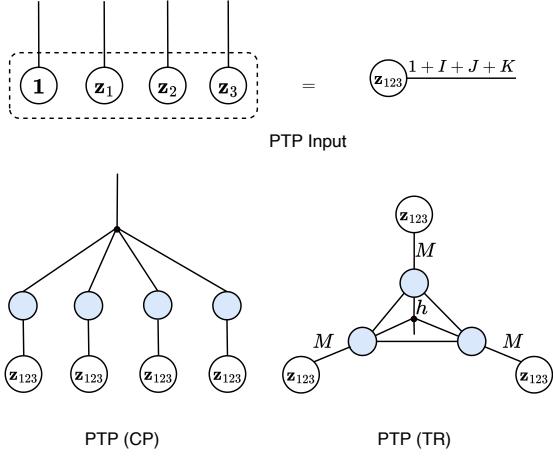


Fig. 9: TN diagrams of PTP, CP and TR structures can be adopted in such a strategy.

quantum states in Section 5.2, and finally introduce the famous quantum TNN model, i.e., ConvAC in Section 5.3.

5.1 Quantum State Embedding for Classic Data

To process machine learning tasks in a quantum system, the input data should be converted into a linear combination of some quantum states as an orthogonal basis:

$$|\psi\rangle = \sum_{d_1 \dots d_N=1}^M \mathcal{A}_{d_1 \dots d_N} |\psi_{d_1}\rangle \circ \dots \circ |\psi_{d_N}\rangle, \quad (34)$$

$$s.t \quad \sum_{d_1 \dots d_N=1}^M \mathcal{A}_{d_1 \dots d_N}^2 = 1, \quad \mathcal{A}_{d_1 \dots d_N} \geq 0,$$

where $|\cdot\rangle$ is the Dirac notation of a vector with complex values [181], and \circ denotes the outer product operation. The tensor \mathcal{A} is the combination coefficient tensor and is always represented and analyzed via a low-rank TN [8]. To embed classic data into a quantum state for adapting quantum systems, Stoudenmire and Schwab [105] proposed a quantum state mapping function $\phi^i(x_i)$ for the i -th pixel x_i in a grayscale image as

$$\phi^i(x_i) = [\cos(\frac{\pi}{2}x_i), \sin(\frac{\pi}{2}x_i)]. \quad (35)$$

The values of pixels are transformed into the range from 0.0 to 1.0 via the mapping function. Furthermore, a full grayscale image \mathbf{x} can be represented as outer products of the mapped quantum states of each pixel:

$$\Phi^{1,2,\dots,N}(\mathbf{x}) = \phi^1(x_1) \circ \phi^2(x_2) \circ \dots \circ \phi^N(x_N), \quad (36)$$

where $\Phi^{1,2,\dots,N}(\mathbf{x}) \in \mathbb{R}^{2 \times 2 \times \dots \times 2}$. Through Eq. (36), it is feasible to associate realistic images with real quantum systems.

For a natural language document, the i -th word $|x_i\rangle$ can also be represented as the sum of orthogonal quantum state bases $|\phi_{h_i}\rangle$ ($h_i = 1, \dots, M$) [106], [107], [108], [182] corresponding to a specific semantic meaning M :

$$|x_i\rangle = \sum_{h_i=1}^M \alpha_{i,h_i} |\phi_{h_i}\rangle, \quad (37)$$

$$s.t \quad \sum_{h_i=1}^M \alpha_{i,h_i}^2 = 1, \quad \alpha_{i,h_i} \geq 0,$$

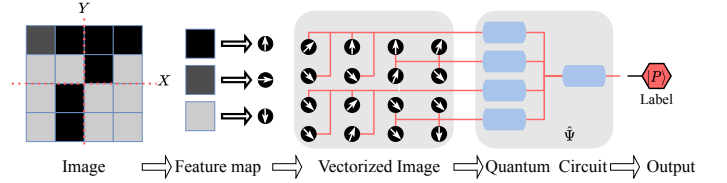


Fig. 10: The processing procedure employed for quantum embedded data [183]. Quantum circuits can be simulated via TNNs on classic electronic computers, and some special TNNs (such as ConvAC) can also be theoretically implemented on a realistic quantum circuit.

where α_{i,h_i} is the associated combination coefficient for each semantic meaning. The constraint of α_i is and . After completing data mapping, the embedded quantum data can be processed by TNNs on a realistic quantum circuit, as shown in Fig. 10. The loss functions of TNNs can also be defined through the properties of quantum circuits. Such a procedure can be simulated on classic electronic computers via TNs and can be theoretically efficiently implemented on realistic quantum systems.

5.2 Embedded Quantum Data Processing

Two series of learning methods are important and have potential for designing and optimizing TNNs, which can be implemented on realistic quantum circuits. One involves applying the density matrix renormalization group (DMRG) algorithm [8], [184] to train supervised models. The other adopts the ideas of the Born machine [185] to learn data distributions via an unsupervised procedure. We introduce these methods in the next part.

5.2.1 Supervised TN Models

Supervised models are used to model the conditional probability distributions of labels (output) given input features based on example input-output pairs. Taking embedded quantum data as inputs, Stoudenmire and Schwab [105] proposed supervised MPS-like tensorial multilinear models and adopted DMRG-like algorithms to optimize the model weights. Prior to introducing a specific implementation, their models must first be formulated as procedures that optimize a set of functions indexed by different labels ℓ :

$$f^\ell(\mathbf{x}) = \mathcal{W}^\ell \times_{1,2,\dots,N} \Phi(\mathbf{x}), \quad (38)$$

where $\Phi(\cdot)$ is the feature map function in Eq. (36) and $\mathcal{W}^\ell \in \mathbb{R}^{2 \times 2 \times \dots \times 2}$ is the weight tensor. Then, the aforementioned tensorial models can be derived by replacing \mathcal{W}^ℓ with an MPS TN:

$$\mathcal{W}_{s_1 s_2 \dots s_N}^\ell = \sum_{\{\alpha\}} \mathcal{A}_{s_1, \alpha_1}^{(1)} \dots \mathcal{A}_{\alpha_{m-1}, s_m, \alpha_m}^{(m)} \dots \mathcal{A}_{\alpha_{N-1}, s_N}^{(N)}, \quad (39)$$

where $\mathcal{A}^{(1)} \dots \mathcal{A}^{(N)}$ are core tensors. Furthermore, the authors proposed an optimization algorithm named Sweeping that was motivated by the DMRG algorithm [8] in quantum mechanics. The optimization algorithm sweeps along an MPS to optimize the quadratic cost function:

$$c = \frac{1}{2} \sum_{n=1}^{N_T} \sum_{\ell} \left(f^\ell(\mathbf{x}_n) - y_{n\ell} \right)^2, \quad (40)$$

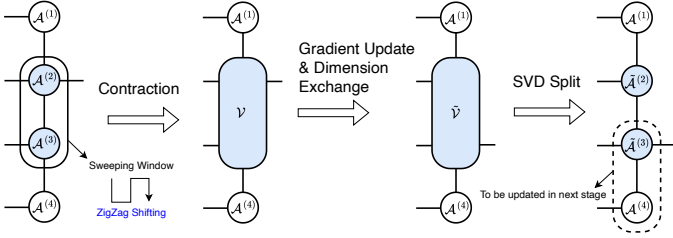


Fig. 11: A single stage of the Sweeping method [105]. In each stage, Sweeping only updates the nodes in the sweeping window, which shifts along a zigzag trajectory.

where N_T denotes the number of training samples, and \mathbf{y}_n denotes the true one-hot label vector of \mathbf{x}_n . The optimization process is carried out to minimize this cost function in stages with stochastic gradient descent. A single stage is shown in Fig. 11. In each stage, two MPS tensors $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(3)}$ are combined into a single bond tensor \mathcal{V} via tensor contraction. Then, the tensor \mathcal{V} is updated with gradients. Finally, $\hat{\mathcal{V}}$ is decomposed back into separate tensors with the SVD algorithm. The Sweeping method is efficient in optimizing models whose inputs embedded quantum data, and it can also be adopted to train TNNs. In addition, other TNs, including the PEPS structure [186], can be processed with a Sweeping-like method.

5.2.2 Unsupervised TN Models

The goal of unsupervised generative modeling is to model the joint probability distribution of the given data. Generative adversarial networks (GANs) [187] and variational autoencoders (VAE) [188] are successful NN models for addressing classic data distributions. Embedded quantum data are the key to training and designing quantum TNNs for generating probabilistic distributions via TNs. Inspired by the probabilistic interpretation of quantum states in quantum mechanics [189], an MPS-based generative model called the Born machine [185] was proposed. The Born machine is an energy-based model [190] derived from quantum mechanics. The distribution functions of the Born machine are shown as follows:

$$P(\mathbf{x}) = \frac{|\Psi(\mathbf{x})|^2}{Z}, \quad (41)$$

where $Z = \sum_{\mathbf{x}} |\Psi(\mathbf{x})|^2$ is the normalization factor, $\Psi(\cdot)$ is the quantum state embedding function, and the energy function of \mathbf{x} can be represented as $|\Psi(\mathbf{x})|^2$ in view of quantum mechanics. $\Psi(\mathbf{x})$ can be parameterized via a TN format [185]. The learning procedure can also be conducted via a DMRG-like algorithm and a gauge transformation strategy [191]. The distribution definitions in Eq. (41) are also useful for designing the loss functions of quantum TNNs. Furthermore, a series of Born machine structures have been proposed, including tree TNs [109], uniform MPSs for language generation [108], and locally purified states [110]. In the future, it is expected that representing and processing data via a particular quantum view will pave the way for the further implementation of TNNs on realistic quantum computers [109].

5.3 ConvAC Network

The expressive power of previously developed quantum data processing models, e.g., the MPS models in Section 5.2.1 and the Born machine in Section 5.2.2, suffers from a lack of nonlinearity. Classic nonlinear operators, e.g., activation functions (such as the rectified linear unit (ReLU) function) and average/max pooling,

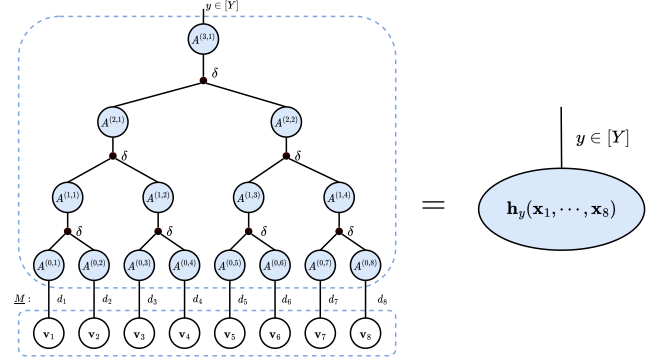


Fig. 12: ConvAC is equivalent to an HT-like TN [180]. $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^s$. \mathbf{x}_1 corresponds to a local patch from the input image or the feature map, and $v^{(0,j)}$ is the linear transformation of \mathbf{x}_j . The width is 2 for a single block. Notably, a single layer is equivalent to a CP format.

can significantly benefit model performance. However, classic nonlinearity cannot be directly implemented in a quantum circuit. To solve this problem, the ConvAC network [111], [192] was proposed to adopt quantum deployable product pooling as a nonlinear operator, proving that ConvAC can be transformed into ConvNets with ReLU activations and average/max pooling.

The whole structure of ConvAC can be represented by an HT format and has been proven to be theoretically deployable in realistic quantum systems. A tensor diagram example of ConvAC is shown in Fig. 12, and one hidden layer of ConvAC is in a CP format. ConvAC can also handle language data [107] by mapping natural language sentences into quantum states via Eq. (37). ConvAC is a milestone in that deep convolutional networks, along with nonlinear modules, are implemented on quantum circuits. It serves as an inspiration for the integration of more NNs into quantum systems. For instance, Zhang et al. [112] introduced the tensor space language model (TSLM), which has been shown to be a generalization of the n-gram language model.

Remark. The implementation of quantum TNNs is a symbolic milestone that forms a bridge between QNNs and classic NNs via TNs. However, strict mapping algorithms between simulated quantum TNNs and realistic physical systems still need to be explored. Moreover, as real high-performance quantum computers are still a long way from being developed, the concept of verifying the performance of quantum TNNs in the near future is infeasible. Despite these issues, it is still important and meaningful to focus on mapping algorithms and performance verifications for QNNs.

6 TRAINING STRATEGIES FOR TNNs

While the aforementioned TNNs can perform well on various tasks and machines, it is also worth exploring training strategies with more stability, better performance and higher efficiency. In this section, we introduce such strategies in three groups; (1) strategies for stabilizing the training processes of TNNs are presented in Section 6.1, (2) strategies for selecting and searching the ranks of TNNs are provided in Section 6.2, and (3) strategies for applying hardware speedup are shown in Section 6.3.

6.1 Stable Training Approaches

Despite a variety of successes, TNNs still suffer from training problems due to their multilinear characteristics. Compared to

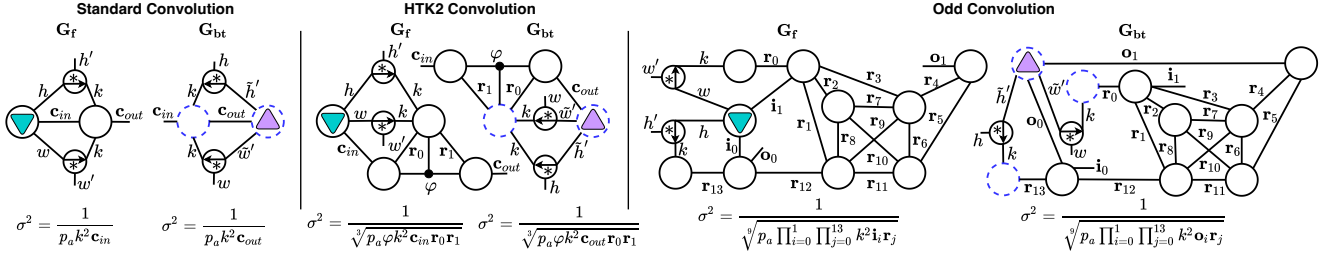


Fig. 13: Three cases of unified TCNN initialization [114]. σ^2 denotes the initial variance of each weight vertex. G_f denotes a forward procedure, and G_b denotes a backward procedure. (i) Standard convolution. The method in [114] degenerates to Xavier/Kaiming initialization on the standard convolution for the same weight variance formulation. (ii) Hyper Tucker-2 (HTK2) convolution. Tucker-2 (TK2) is a common TD that is utilized in ResNet as the bottleneck module [193]. HTK2 is formed by applying a hyperedge to the weight vertices of TK2. (iii) Odd convolution. The odd TD was originally proposed by [194]. The connections among the vertices are irregular, making weight initialization a complex problem. These three successful initialization cases can better demonstrate the potential adaptability of unified initialization to diverse TCNNs.

simple linear operations such as matrix production, tensor contraction yields data flows with exponential scales, i.e., features in forward propagation and gradients in backward propagation, when modes linearly increase [114]. One solution is to utilize the full-precision float64 format to denote large weights, which can alleviate these numerical issues to some extent. However, a full-precision format can result in more calculations and higher time consumption compared to a lower-precision format, e.g., float16. Nevertheless, low precision may cause numerical stability issues. To solve these problems, Panagakis et al. [113] proposed a mixed-precision strategy to form a tradeoff. This dynamic precision strategy is efficient in reducing memory occupation and promotes training stability.

Another feasible way to solve the training problem lies in developing a suitable initialization. Commonly used adaptive initialization methods include Xavier [195] initialization and Kaiming [196] initialization, which regulate the variances of the data flows in layers. However, these two initializations cannot calculate the correct scales of TNNs to neglect interactions in tensor contractions. Furthermore, tensor formats are different from each other, causing difficulty in developing a general fitting initialization for diverse tensorial layers. To solve these two problems, Yu initialization [114] proposed a unified initialization paradigm based on Xavier to adaptively initialize for arbitrary TCNNs. Specifically, Pan et al. extracted a backbone graph (BG) from a tensorial convolution hypergraph [61] and then encoded an arbitrary TCNN into an adjacency matrix with this BG. Finally, a suitable initial variance for a TCNN can be directly calculated through the adjacency matrix. We illustrate three cases of using unified initializations in Fig. 13. Although Yu initialization was proposed for TCNNs, it can also be widely used in other NNs, including CNNs, RNNs, and Transformers, since these models basic layers, i.e., convolutional and linear layers, belong to the scope of TCNNs.

6.2 Rank Selection and Search

Prior studies [67], [69], [79] focused on finding efficient TN formats (e.g., TTs and TRs) for compressing NNs and achieve significant efficiency for their natural compact structures. However, despite these remarkable successes, efficient algorithms for adjusting or selecting suitable ranks for a TN are lacking since rank selection is an NP-hard problem [153]. As a result, many approaches [64], [68], [69], [77] can only set values for all ranks manually, which severely affects the resulting models' training procedures. Fortunately, the

rank selection problem can still be optimized through heuristic strategies, such as Bayesian optimization [118], reinforcement learning (RL) [116] and evolutionary algorithms (EAs) [115]. Here, we introduce some rank selection methods for TNNs.

DNNs utilize neural architecture search (NAS) [197] to search for the optimal network hyperparameters, achieving significant success. As ranks can be treated as architecture hyperparameters, NAS is applicable to searching for optimal tensorial layers with better rank settings. Following this idea, the progressive searching TR network (PSTRN) [115] employs NAS with an EA to select suitable ranks for a TR network (TRN). In detail, the PSTRN employs a heuristic hypothesis for searching: “when a shape-fixed TRN performs well, part or all of its rank elements are sensitive, and each of them tends to aggregate in a narrow region, which is called an interest region”. Instructed by the interest region hypothesis, the PSTRN can reach the optimal point with a higher probability than a plain EA method. The PSTRN consists of an evolutionary phase and a progressive phase. During the evolutionary phase, this method validates the ranks in the search space on benchmarks and picks the rank that yields the best performance. Then, in the progressive phase, the PSTRN samples new ranks around the previously picked rank and inserts them into a new search space. After several rounds, the heuristic EA can find a high-performance solution. With such an efficient design, the PSTRN successfully achieves better performance than hand-setting, which demonstrates that its hypothesis is practical.

In addition to NAS, some other efficient methods are also available for rank selection. Zhao et al. [117] inferred a CP rank by implementing a reduction process on a large rank value via a variational Bayesian optimization procedure. Hawkins and Zhang [118] extended this CP procedure [117] to TT-based TNNs and adopted the Stein variational gradient descent method, which combines the flexibility of the Markov chain Monte Carlo (MCMC) approach with the speed of variational Bayesian inference to construct a Bayesian optimization method. In pretrained networks, Kim et al. [120] and Gusak et al. [121] derive approximate ranks by employing Bayesian matrix factorization (BMF) [198] to unfolding weight tensors. Unlike Bayesian methods, Cheng et al. [116] treated the rank searching task as a game process whose search space was irregular, thus applying RL to find comparably suitable ranks for a trained CNN. However, this algorithm is TD-dependent, which indicates that its performance may be influenced by the selected TD method. Yin et al. [119] leveraged the alternating direction

method of multipliers (ADMM) to gradually transfer the original weight to a low-rank representation (i.e., a TT).

6.3 Hardware Speedup

Accelerating the training and inference procedures of TNNs can benefit resource consumption and experimental adjustment, thereby achieving economic gains and green research. A direct and effective approach is to optimize the speed of tensor operations in TNNs to realize hardware acceleration. As inferring TT-format TNNs inevitably results in enormous quantities of redundant calculations, the TIE scheme [122] was proposed to accelerate TT layers by splitting the working SRAM into numerous groups with a well-designed data selection mechanism. Huang et al. [123] designed a parallel computation scheme with higher I/O bandwidth, improving the speed of tensor contractions. Later, they proposed an LTNN [123] to map TT-format TNNs into a 3D accelerator based on CMOS-RRAM, leading to significantly increased bandwidth via vertical I/O connections. As a result, they simultaneously attained high throughput and low power consumption for TNNs. Recently, Qu et al. [124] proposed a spatial 2D processing element (PE) array architecture and built a hardware TT engine consisting of off-chip DRAM. Kao et al. [125] proposed an energy-efficient hardware accelerator for CP convolution with a mixing method that combines the Walsh-Hadamard transform and the discrete cosine transform.

Many more fascinating methods have been developed for the acceleration of generic tensor operations, which are correlated with TNNs. For instance, Huang et al. [199] observed that the tensor matricization operation is usually resource-consuming since its DRAM access is built on a random reading address; thus, they proposed a tensor storage scheme with a sequential address design for better DRAM accessibility. Both T2s-tensor [200] and Tensaurus [201] mainly focus on designing general computation kernels for dense and sparse tensor data. Xie et al. [149] and Liang et al. [202] accelerated search procedures for obtaining an optimal sequence of tensor contractions. Xie et al. [149] solved the massive computational complexity problem of double-layer TN contraction in quantum analysis and mapped such a double-layer TN onto an intersected single-layer TN. Liang et al. [202] implemented multithread optimization to improve the parallelism of contractions. Fawzi et al. [203] also illustrated the potential of RL to build efficient universal tensor operations. In the future, it is expected that more general hardware acceleration schemes based on tensor operations will be developed to implement TNNs with smaller storage and time consumption levels.

Remark. The comments are divided into three parts. (1) To achieve training stability, it is possible to borrow ideas concerning identity transition maintenance to construct more stable initializations. In addition, it is also feasible to add adversarial examples to enhance network robustness. (2) Rank search is important for further improving the performance of TNNs. However, as it is an NP-hard problem, rank search has not been sufficiently explored. In the future, suitable ranks can be searched through the guidance of gradient sizes and EAs in searching for TNN architectures. (3) Last, research on hardware has derived some success in terms of speed acceleration and memory reduction. However, these methods are almost ad hoc designs for specific TD formats, so they lack applicability to other TNN structures.

7 TNN TOOLBOXES

In 1973, Pereyra and Scherer [204], as pioneers in this field, developed a programming technique for basic tensor operations.

Recently, with the development of modern computers, many more basic tensor operation toolboxes have been developed, and a series of powerful TNN toolboxes have also been proposed for both network compression and quantum circuit simulation, which are the two main applications of TNNs. In this section, toolboxes for TNNs are presented in three categories according to their design purposes: (1) toolboxes for basic tensor operations contain important and fundamental operations (e.g., tensor contraction and permutation) in TNNs (Section 7.1); (2) toolboxes for network compression are high-level TNN architecture toolboxes based on other basic operation tools (Section 7.2); and (3) toolboxes for quantum circuit simulation are software packages for the quantum circuit simulation or quantum machine learning processes that use TNs from a quantum perspective (Section 7.3).

7.1 Toolboxes for Basic Tensor Operations

Toolboxes for basic tensor operations aim to implement some specific TD algorithms. Many basic tensor toolboxes based on different programming languages and backends have been designed for this purpose. For example, The online stochastic framework for TD (OSTD) [130] and Tensor Toolbox [128] were constructed for low-rank decomposition and implemented with MATLAB. Regarding Python-based toolboxes, TensorTools based on NumPy [205] implements CP only, while T3F [136] was explicitly designed for TT decomposition on TensorFlow [206]. Similarly, based on TensorFlow, TensorD [131] supports CP and Tucker decomposition. Tntorch [133] is a PyTorch-based library for tensor modeling in the CP, Tucker and TT formats. TorchMPS [134], TT-Toolbox [132] and Scikit-TT [138] are all powerful Python-based specific TT solvers that efficiently implement the DMRG algorithm. Tensorly is a powerful general TD library that supports many decomposition formats and various Python backends including CuPy, Pytorch, TensorFlow and MXNet [207]. TensorNetwork [137] is a powerful general-purpose TN library that supports a variety of Python backends, including JAX, TensorFlow, PyTorch and NumPy. In addition, some toolboxes based on C++ are also available. TenDeC++ [208] leverages a unique pointer technology called PointerDeformer in C++ to support the efficient computation of TD functions. ITensor [135] is an efficient and flexible C++ library for general TN calculations.

7.2 Toolboxes for Network Compression

Specific TNN toolboxes are used to assist with the development of tensorial layers. Although some general tensor toolboxes such as Tensorly [126] are powerful for TD processing and can use their TD operations to help initialize TNN modules to a certain extent, they still lack support for application programming interfaces (APIs) for building TNNs directly. Therefore, a TNN library (Tensorly-Torch) based on Tensorly was developed to build some tensor layers within any PyTorch network. Pan et al. also developed a powerful TNN library called TedNet [64]. TedNet can quickly set up TNN layers by directly calling the API. In addition, TedNet supports the construction of TCNNs and TRNNs in single lines of code.

7.3 Toolboxes for Quantum Circuit Simulation

A number of quantum circuit simulation toolboxes have been designed. For example, some TT toolboxes such as Scikit-TT and TorchMPS can partially simulate quantum circuits to some extent, although they were not specifically designed for quantum circuit

simulation. In contrast, general TN toolboxes, e.g., TensorNetwork and ITensor, can simulate any quantum circuit. In addition, with optimized tensor contraction, TeD-Q [141], a TN-enhanced open-source software framework for quantum machine learning, enables the simulation of large quantum circuits. Furthermore, Yao [139], an extensible and efficient library for designing quantum algorithms, can provide support for dumping a quantum circuit into a TN. Although no practical implementations of quantum TNNs are available, these quantum circuit simulations are potentially useful for the simulation of quantum TNNs.

Remark. Despite the success of current toolboxes, some areas for improvement remain. (1) Existing basic tensor operation toolboxes are built using high-level software frameworks, limiting their ability to fully utilize the inherent capability of tensor computations. (2) Existing deep model implementation toolboxes for TNNs can only contain a limited number of predefined TNN structures and cannot allow users to design structures freely. (3) Existing quantum simulation toolboxes focus more on the simulation of quantum circuits using TNs and do not facilitate the processing of embedded quantum data via TNNs.

8 CONCLUSION AND FUTURE PROSPECTS

This survey embraces the connection between TNs and NNs, summarizing the techniques for compressing NN parameters, modeling the interactions between different dimensions of data, and bridging QNNs with classic NNs. As previously noted, TNNs have considerable strengths and immense potential, making them applicable in many areas. In the future, research on TNNs can benefit from the optimization and implementation of tensor-friendly hardware; we believe that TNNs will have large impacts on studies involving quantum physics and quantum computers.

Acceleration based on hardware design. Although many TNNs have low calculation complexity levels in theory, realistic hardware deployments usually fall short of this objective due to their numerous permutation operations [64] and the absence of sufficient parallelism [123]. Efficient hardware and matching software for universal tensor acceleration or TNN acceleration can be developed. As mentioned in Section 6.3, the existing tensor acceleration software and hardware structures are all aimed at certain TN structures or are based on parallel matrix acceleration. The acceleration of general TN operations is necessary and urgent for the implementation of TNNs.

Applications in quantum physics. In some specific physical applications that need to deal with large-scale tensors, such as wave function simulation [209], specifically designed TNNs can be studied to efficiently solve these problems involving higher-order interactions. Inspired by the universal approximation theorem, some simple NNs have been adopted in wave function simulation tasks, such as free boson and fermion systems [210]. However, because of the curse of dimensionality, simple NNs are difficult to apply in extremely large-scale wave function simulation tasks, and TNNs can easily handle tasks with large-scale tensor as a result of the compact nature of TNs.

Implementations in quantum mechanics. The existing TNNs mainly adopt the mathematical forms of TNs and seldom consider the physical properties of the quantum systems described by these TNs [12], [113]. Since TNNs are highly related to quantum circuit structures, it is possible to obtain more efficient and effective TNNs by applying the concepts and theory of TNs in quantum mechanics. For example, optimizing and interpreting TNNs from

the perspective of entanglement entropy theory [211] could be a meaningful direction for producing interpretable and efficient NNs.

ACKNOWLEDGMENTS

This paper was partially supported by National Key Research and Development Program of China (No. 2018AAA0100204), a key program of fundamental research from Shenzhen Science and Technology Innovation Commission (No. JCYJ20200109113403826), and the Major Key Project of PCL (No. PCL2021A06).

REFERENCES

- [1] D. Cyganski and J. A. Orr, "Applications of tensor theory to object recognition and orientation determination," *IEEE Trans. PAMI*, no. 6, pp. 662–673, 1985.
- [2] P. Koniusz, L. Wang, and A. Cherian, "Tensor representations for action recognition," *IEEE Trans. PAMI*, vol. 44, no. 2, pp. 648–665, 2021.
- [3] J. Tang, X. Shu, Z. Li, Y.-G. Jiang, and Q. Tian, "Social anchor-unit graph regularized tensor completion for large-scale image retagging," *IEEE Trans. PAMI*, vol. 41, no. 8, pp. 2027–2034, 2019.
- [4] J. Tang, X. Shu, G.-J. Qi, Z. Li, M. Wang, S. Yan, and R. Jain, "Tri-clustered tensor completion for social-aware image tag refinement," *IEEE Trans. PAMI*, vol. 39, no. 8, pp. 1662–1674, 2016.
- [5] I. Davidson, S. Gilpin, O. Carmichael, and P. Walker, "Network discovery via constrained tensor analysis of fmri data," in *ACM SIGKDD*, 2013.
- [6] E. Acar, Y. Levin-Schwartz, V. D. Calhoun, and T. Adali, "Tensor-based fusion of EEG and fMRI to understand neurological changes in schizophrenia," in *ISCAS*. IEEE, 2017.
- [7] K. Keegan, T. Vishwanath, and Y. Xu, "A tensor SVD-based classification algorithm applied to fMRI data," *arXiv preprint arXiv:2111.00587*, 2021.
- [8] J. Biamonte and V. Bergholm, "Tensor networks in a nutshell," *arXiv preprint arXiv:1708.00006*, 2017.
- [9] T. Huckle, K. Waldherr, and T. Schulte-Herbrüggen, "Computations in quantum tensor networks," *Linear Algebra and its Applications*, vol. 438, no. 2, pp. 750–781, 2013.
- [10] X. Chen, Z. He, and L. Sun, "A bayesian tensor decomposition approach for spatiotemporal traffic data imputation," *Transportation research part C: emerging technologies*, vol. 98, pp. 73–84, 2019.
- [11] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, D. P. Mandic *et al.*, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Foundations and Trends® in Machine Learning*, vol. 9, no. 4-5, pp. 249–429, 2016.
- [12] A. Cichocki, A.-H. Phan, Q. Zhao, N. Lee, I. Oseledets, M. Sugiyama, D. P. Mandic *et al.*, "Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives," *Foundations and Trends® in Machine Learning*, vol. 9, no. 6, pp. 431–673, 2017.
- [13] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.
- [14] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.
- [15] H. A. Kiers, "Towards a standardized notation and terminology in multi-way analysis," *Journal of Chemometrics: A Journal of the Chemometrics Society*, vol. 14, no. 3, pp. 105–122, 2000.
- [16] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.
- [17] —, "Implications of factor analysis of three-way matrices for measurement of change," *Problems in measuring change*, vol. 15, pp. 122–137, 1963.
- [18] L. De Lathauwer, "Decompositions of a higher-order tensor in block terms—part I: Lemmas for partitioned matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1022–1032, 2008.
- [19] —, "Decompositions of a higher-order tensor in block terms—part II: Definitions and uniqueness," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1033–1066, 2008.
- [20] L. De Lathauwer and D. Nion, "Decompositions of a higher-order tensor in block terms—part III: Alternating least squares algorithms," *SIAM journal on Matrix Analysis and Applications*, vol. 30, no. 3, pp. 1067–1083, 2008.
- [21] I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.

- [22] A. Cichocki, "Era of big data processing: A new approach via tensor networks and tensor decompositions," *arXiv preprint arXiv:1403.2048*, 2014.
- [23] D. Perez-García, F. Verstraete, M. M. Wolf, and J. I. Cirac, "Matrix product state representations," *Quantum Information and Computation*, vol. 7, no. 5-6, pp. 401–430, 2007.
- [24] F. Verstraete, V. Murg, and J. I. Cirac, "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems," *Advances in Physics*, vol. 57, no. 2, pp. 143–224, 2008.
- [25] Q. Zhao, G. Zhou, S. Xie, L. Zhang, and A. Cichocki, "Tensor ring decomposition," *arXiv preprint arXiv:1606.05535*, 2016.
- [26] D. Kressner and C. Tobler, "Htucker—a MATLAB toolbox for tensors in hierarchical Tucker format," *Mathicse, EPF Lausanne*, 2012.
- [27] F. Verstraete and J. I. Cirac, "Renormalization algorithms for quantum-many body systems in two and higher dimensions," *arXiv preprint cond-mat/0407066*, 2004.
- [28] N. Schuch, M. M. Wolf, F. Verstraete, and J. I. Cirac, "Computational complexity of projected entangled pair states," *Physical review letters*, vol. 98, no. 14, p. 140506, 2007.
- [29] A. Milsted and G. Vidal, "Geometric interpretation of the multi-scale entanglement renormalization ansatz," *arXiv preprint arXiv:1812.00529*, 2018.
- [30] F. Pan and P. Zhang, "Simulation of quantum circuits using the big-batch tensor network method," *Physical Review Letters*, vol. 128, no. 3, p. 030501, 2022.
- [31] Z. Zhang, G. I. Allen, H. Zhu, and D. Dunson, "Tensor network factorizations: Relationships between brain structural connectomes and traits," *Neuroimage*, vol. 197, pp. 330–343, 2019.
- [32] S. Sharma and A. Alavi, "Multireference linearized coupled cluster theory for strongly correlated systems using matrix product states," *The Journal of chemical physics*, vol. 143, no. 10, p. 102815, 2015.
- [33] X. Cao, X. Wei, Y. Han, and D. Lin, "Robust face clustering via tensor decomposition," *IEEE transactions on cybernetics*, vol. 45, no. 11, pp. 2546–2557, 2014.
- [34] A. Zare, A. Ozdemir, M. A. Iwen, and S. Aviyente, "Extension of pca to higher order data structures: An introduction to tensors, tensor decompositions, and tensor PCA," *Proceedings of the IEEE*, vol. 106, no. 8, pp. 1341–1358, 2018.
- [35] J. Liu, P. Musialski, P. Wonka, and J. Ye, "Tensor completion for estimating missing values in visual data," *IEEE Trans. PAMI*, vol. 35, no. 1, pp. 208–220, 2012.
- [36] Z. Xu, F. Yan, and Y. Qi, "Bayesian nonparametric models for multiway data analysis," *IEEE Trans. PAMI*, vol. 37, no. 2, pp. 475–487, 2015.
- [37] J. Zhang, X. Li, P. Jing, J. Liu, and Y. Su, "Low-rank regularized heterogeneous tensor decomposition for subspace clustering," *IEEE Signal Processing Letters*, vol. 25, no. 3, pp. 333–337, 2017.
- [38] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.
- [39] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.
- [40] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.
- [41] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.
- [42] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.
- [43] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017.
- [44] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," in *ACL*, 2016.
- [45] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (GRU) neural networks," in *MWSCAS*, 2017.
- [46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, E. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.
- [47] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.
- [48] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2020.
- [49] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.
- [50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NeurIPS*, 2012.
- [51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.
- [53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
- [54] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [55] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Židek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [56] M. van Breugel, I. Rosa e Silva, and A. Andreeva, "Structural validation and assessment of AlphaFold2 predictions for centrosomal and centriolar proteins and their complexes," *Communications Biology*, vol. 5, no. 1, pp. 1–10, 2022.
- [57] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, "Strategies for training large scale neural network language models," in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2011, pp. 196–201.
- [58] M. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, no. 12, pp. i121–i129, 2014.
- [59] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery," *Drug discovery today*, vol. 23, no. 6, pp. 1241–1250, 2018.
- [60] Y. Zhou, E. Lentz, H. Michelson, C. Kim, and K. Baylis, "Machine learning for food security: Principles for transparency and usability," *Applied Economic Perspectives and Policy*, vol. 44, no. 2, pp. 893–910, 2022.
- [61] K. Hayashi, T. Yamaguchi, Y. Sugawara, and S. Maeda, "Exploring unexplored tensor network decompositions for convolutional neural networks," in *NeurIPS*, 2019.
- [62] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," in *ICLR*, 2015.
- [63] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NeurIPS*, 2014, pp. 1269–1277.
- [64] Y. Pan, M. Wang, and Z. Xu, "Tednet: A Pytorch toolkit for tensor decomposition networks," *Neurocomputing*, vol. 469, pp. 234–238, 2022.
- [65] Y. Liu and M. K. Ng, "Deep neural network compression by Tucker decomposition with nonlinear response," *Knowledge-Based Systems*, 2022.
- [66] T. Garipov, D. Podoprikin, A. Novikov, and D. Vetrov, "Ultimate tensorization: compressing convolutional and fc layers alike," *arXiv preprint arXiv:1611.03214*, 2016.
- [67] A. Novikov, D. Podoprikin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *NeurIPS*, 2015.
- [68] W. Wang, Y. Sun, B. Eriksson, W. Wang, and V. Aggarwal, "Wide compression: Tensor ring nets," in *CVPR*, 2018, pp. 9329–9338.
- [69] J. Ye, G. Li, D. Chen, H. Yang, S. Zhe, and Z. Xu, "Block-term tensor neural networks," *Neural Networks: the Official Journal of the International Neural Network Society*, vol. 130, pp. 11–21, 2020.
- [70] J. Kossai, A. Khanna, Z. Lipton, T. Furlanello, and A. Anandkumar, "Tensor contraction layers for parsimonious deep nets," in *CVPR Workshops*, 2017.
- [71] J. Kossai, Z. C. Lipton, A. Kolbeinsson, A. Khanna, T. Furlanello, and A. Anandkumar, "Tensor regression networks," *J. Mach. Learn. Res.*, vol. 21, no. 123, pp. 1–21, 2020.
- [72] B. Wu, D. Wang, G. Zhao, L. Deng, and G. Li, "Hybrid tensor decomposition in neural network compression," *Neural Networks*, vol. 132, pp. 309–320, 2020.
- [73] J. Kossai, A. Bulat, G. Tzimiropoulos, and M. Pantic, "T-net: Parametrizing fully convolutional nets with a single high-order tensor," in *CVPR*, 2019.
- [74] Y. Yang and T. Hospedales, "Deep multi-task representation learning: A tensor factorisation approach," in *ICLR*, 2017.

- [75] M. Wang, Z. Su, X. Luo, Y. Pan, S. Zheng, and Z. Xu, "Concatenated tensor networks for deep multi-task learning," in *ICONIP*, 2020.
- [76] J. Kossaifi, A. Toisoul, A. Bulat, Y. Panagakis, T. M. Hospedales, and M. Pantic, "Factorized higher-order cnns with an application to spatio-temporal emotion estimation," in *CVPR*, 2020.
- [77] Y. Yang, D. Krompass, and V. Tresp, "Tensor-Train recurrent neural networks for video classification," in *ICML*, 2017.
- [78] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu, "Learning compact recurrent neural networks with block-term tensor decomposition," in *CVPR*, 2018.
- [79] Y. Pan, J. Xu, M. Wang, J. Ye, F. Wang, K. Bai, and Z. Xu, "Compressing recurrent neural networks with tensor ring for action recognition," in *AAAI*, 2019.
- [80] M. Yin, S. Liao, X. Liu, X. Wang, and B. Yuan, "Towards extremely compact rnns for video recognition with fully decomposed hierarchical tucker structure," in *CVPR*, 2021.
- [81] C. Jose, M. Cissé, and F. Fleuret, "Kronecker recurrent units," in *ICML*, 2018.
- [82] A. Tjandra, S. Sakti, and S. Nakamura, "Recurrent neural network compression based on low-rank tensor representation," *IEICE Trans. Inf. Syst.*, vol. 103-D, no. 2, pp. 435–449, 2020.
- [83] D. Wang, B. Wu, G.-S. Zhao, H. Chen, L. Deng, T. Yan, and G. Li, "Kronecker cp decomposition with fast multiplication for compressing rnns," *IEEE Trans. NNLS*, vol. PP, 2021.
- [84] J. Su, W. Byeon, J. Kossaifi, F. Huang, J. Kautz, and A. Anandkumar, "Convolutional tensor-train LSTM for spatio-temporal learning," in *NeurIPS*, 2020.
- [85] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, M. Zhou, and D. Song, "A tensorized transformer for language modeling," *NeurIPS*, 2019.
- [86] P. Liu, Z. Gao, W. X. Zhao, Z. Xie, Z. Lu, and J. Wen, "Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators," in *ACL/IJCNLP*, 2021.
- [87] X. Liu, J. Su, and F. Huang, "Tuformer: Data-driven design of expressive transformer by Tucker tensor representation," in *ICLR*, 2022.
- [88] B. Wang, Y. Ren, L. Shang, X. Jiang, and Q. Liu, "Exploring extreme parameter compression for pre-trained language models," in *ICLR*, 2022.
- [89] L. Sunzhu, Z. Peng, G. Guobing, L. Xiuqing, W. Benyou, W. Junqiu, and J. Xin, "Hypoformer: Hybrid decomposition transformer for edge-friendly neural machine translation," *EMNLP*, 2022.
- [90] C. Hua, G. Rabusseau, and J. Tang, "High-order pooling for graph neural networks with tensor decomposition," *NeurIPS*, 2022.
- [91] P. Baghershahi, R. Hosseini, and H. Moradi, "Efficient relation-aware neighborhood aggregation in graph neural networks via tensor decomposition," *arXiv preprint arXiv:2212.05581*, 2022.
- [92] C. Jia, B. Wu, and X.-P. Zhang, "Dynamic spatiotemporal graph neural network with tensor network," *arXiv preprint arXiv:2003.08729*, 2020.
- [93] F. Ju, Y. Sun, J. Gao, M. Antolovich, J. Dong, and B. Yin, "Tensorizing restricted Boltzmann machine," *ACM Transactions on Knowledge Discovery from Data*, vol. 13, no. 3, pp. 1–16, 2019.
- [94] M. Wang, C. Zhang, Y. Pan, J. Xu, and Z. Xu, "Tensor ring restricted Boltzmann machines," in *IJCNN*, 2019.
- [95] T. D. Nguyen, T. Tran, D. Phung, and S. Venkatesh, "Tensor-variate restricted Boltzmann machines," in *AAAI*, 2015.
- [96] G. Qi, Y. Sun, J. Gao, Y. Hu, and J. Li, "Matrix variate restricted Boltzmann machine," in *IJCNN*, 2016.
- [97] A. Zadeh, M. Chen, S. Poria, E. Cambria, and L.-P. Morency, "Tensor fusion network for multimodal sentiment analysis," in *EMNLP*, 2017.
- [98] S. Rendle, "Factorization machines," in *ICDM*, 2010.
- [99] Z. Liu, Y. Shen, V. B. Lakshminarasimhan, P. P. Liang, A. B. Zadeh, and L.-P. Morency, "Efficient low-rank multimodal fusion with modality-specific factors," in *ACL*, 2018.
- [100] M. Hou, J. Tang, J. Zhang, W. Kong, and Q. Zhao, "Deep multimodal multilinear fusion with high-order polynomial pooling," *NeurIPS*, 2019.
- [101] H. Ben-Younes, R. Cadene, M. Cord, and N. Thome, "Mutan: Multimodal Tucker fusion for visual question answering," in *ICCV*, 2017.
- [102] A. Fukui, D. H. Park, D. Yang, A. Rohrbach, T. Darrell, and M. Rohrbach, "Multimodal compact bilinear pooling for visual question answering and visual grounding," in *EMNLP*, 2016.
- [103] J.-H. Kim, K.-W. On, W. Lim, J. Kim, J.-W. Ha, and B.-T. Zhang, "Hadamard product for low-rank bilinear pooling," *arXiv preprint arXiv:1610.04325*, 2016.
- [104] T. Do, T.-T. Do, H. Tran, E. Tjiputra, and Q. D. Tran, "Compact trilinear interaction for visual question answering," in *ICCV*, 2019.
- [105] E. Stoudenmire and D. J. Schwab, "Supervised learning with tensor networks," *NeurIPS*, 2016.
- [106] Q. Li, B. Wang, and M. Melucci, "CNM: An interpretable complex-valued network for matching," in *NAACL*, 2019.
- [107] P. Zhang, Z. Su, L. Zhang, B. Wang, and D. Song, "A quantum many-body wave function inspired language modeling approach," in *CIKM*, 2018.
- [108] J. Miller, G. Rabusseau, and J. Terilla, "Tensor networks for probabilistic sequence modeling," in *AISTATS*, 2021.
- [109] S. Cheng, L. Wang, T. Xiang, and P. Zhang, "Tree tensor networks for generative modeling," *Physical Review B*, vol. 99, no. 15, p. 155131, 2019.
- [110] I. Glasser, R. Sweke, N. Pancotti, J. Eisert, and I. Cirac, "Expressive power of tensor-network factorizations for probabilistic modeling," *NeurIPS*, 2019.
- [111] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," in *COLT*, 2016.
- [112] L. Zhang, P. Zhang, X. Ma, S. Gu, Z. Su, and D. Song, "A generalized language model in tensor space," in *AAAI*, 2019.
- [113] Y. Panagakis, J. Kossaifi, G. G. Chrysos, J. Oldfield, M. A. Nicolaou, A. Anandkumar, and S. Zafeiriou, "Tensor methods in computer vision and deep learning," *Proc. IEEE*, vol. 109, no. 5, pp. 863–890, 2021.
- [114] Y. Pan, Z. Su, A. Liu, J. Wang, N. Li, and Z. Xu, "A unified weight initialization paradigm for tensorial convolutional neural networks," in *ICML*, 2022.
- [115] N. Li, Y. Pan, Y. Chen, Z. Ding, D. Zhao, and Z. Xu, "Heuristic rank selection with progressively searching tensor ring network," *Complex & Intelligent Systems*, pp. 1–15, 2021.
- [116] Z. Cheng, B. Li, Y. Fan, and Y. Bao, "A novel rank selection scheme in tensor ring decomposition based on reinforcement learning for deep neural networks," in *ICASSP*, 2020.
- [117] Q. Zhao, L. Zhang, and A. Cichocki, "Bayesian CP factorization of incomplete tensors with automatic rank determination," *IEEE Trans. PAMI*, vol. 37, no. 9, pp. 1751–1763, 2015.
- [118] C. Hawkins and Z. Zhang, "Bayesian tensorized neural networks with automatic rank selection," *Neurocomputing*, vol. 453, pp. 172–180, 2021.
- [119] M. Yin, Y. Sui, S. Liao, and B. Yuan, "Towards efficient tensor decomposition-based DNN model compression with optimization framework," in *CVPR*, 2021.
- [120] Y. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR*, 2016.
- [121] J. Gusak, M. Kholyavchenko, E. Ponomarev, L. Markeeva, P. Blagoveschensky, A. Cichocki, and I. V. Oseledets, "Automated multi-stage compression of neural networks," in *ICCV Workshops*, 2019.
- [122] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "TIE: energy-efficient tensor train-based inference engine for deep neural network," in *ISCA*, 2019.
- [123] H. Huang, L. Ni, and H. Yu, "LTNN: An energy-efficient machine learning accelerator on 3d cmos-rram for layer-wise tensorized neural network," in *SOCC*, 2017.
- [124] Z. Qu, L. Deng, B. Wang, H. Chen, J. Lin, L. Liang, G. Li, Z. Zhang, and Y. Xie, "Hardware-enabled efficient data processing with tensor-train decomposition," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 2, pp. 372–385, 2022.
- [125] C. Kao, Y. Hsieh, C. Chen, and C. Yang, "Hardware acceleration in large-scale tensor decomposition for neural network compression," in *MWSCAS*, 2022.
- [126] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in python," *J. Mach. Learn. Res.*, vol. 20, pp. 26:1–26:6, 2019.
- [127] A. H. Williams, T. H. Kim, F. Wang, S. Vyas, S. I. Ryu, K. V. Shenoy, M. Schnitzer, T. G. Kolda, and S. Ganguli, "Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis," *Neuron*, vol. 98, no. 6, pp. 1099–1115, 2018.
- [128] T. G. Kolda and B. W. Bader, "Matlab tensor toolbox," Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA, Tech. Rep., 2006.
- [129] J. Huang, L. Kong, X. Liu, W. Qu, and G. Chen, "A C++ library for tensor decomposition," in *IPCCC*, 2019.
- [130] A. Sobral, S. Javed, S. K. Jung, T. Bouwmans, and E. Zahzah, "Online stochastic tensor decomposition for background subtraction in multispectral video sequences," in *ICCV Workshops*, 2015.
- [131] L. Hao, S. Liang, J. Ye, and Z. Xu, "TensorD: A tensor decomposition library in tensorflow," *Neurocomputing*, vol. 318, pp. 196–200, 2018.
- [132] I. Oseledets, S. Dolgov, V. Kazeev, D. Savostyanov, O. Lebedeva, P. Zhlobich, T. Mach, and L. Song, "TT-toolbox," 2016.
- [133] R. Ballester-Ripoll, "tntorch - tensor network learning with PyTorch," 2018. [Online]. Available: <https://github.com/rballester/tntorch>

- [134] J. Miller, “TorchMPS,” 2019. [Online]. Available: <https://github.com/jemisjoky/torchmps>
- [135] M. Fishman, S. R. White, and E. M. Stoudenmire, “The ITensor software library for tensor network calculations,” 2020.
- [136] A. Novikov, P. Izmailov, V. Khurikov, M. Figurnov, and I. V. Oseledets, “Tensor train decomposition on tensorflow (T3F),” *J. Mach. Learn. Res.*, vol. 21, pp. 30:1–30:7, 2020.
- [137] C. Roberts, A. Milsted, M. Ganahl, A. Zalcman, B. Fontaine, Y. Zou, J. Hidary, G. Vidal, and S. Leichenauer, “Tensornetwork: A library for physics and machine learning,” *arXiv preprint arXiv:1905.01330*, 2019, 2019.
- [138] P. Gel, S. Klus, M. Scherer, and F. Nsae, “Scikit-TT tensor train toolbox,” 2018. [Online]. Available: <https://github.com/PGelss/scikitst>
- [139] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang, “Yao. jl: Extensible, efficient framework for quantum algorithm design,” *Quantum*, vol. 4, p. 341, 2020.
- [140] D. Kartsaklis, I. Fan, R. Yeung, A. Pearson, R. Lorenz, A. Toumi, G. de Felice, K. Meichanetzidis, S. Clark, and B. Coecke, “lambeq: An efficient high-level python library for quantum nlp,” *arXiv preprint arXiv:2110.04236*, 2021.
- [141] J. E. Academy, “Tensor-network enhanced distributed quantum,” 2022. [Online]. Available: <https://github.com/JDEA-Quantum-Lab/TeD-Q>
- [142] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, “Vqa: Visual question answering,” in *ICCV*, 2015.
- [143] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, “The power of quantum neural networks,” *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.
- [144] S. C. Kak, “Quantum neural computing,” *Advances in imaging and electron physics*, vol. 94, pp. 259–313, 1995.
- [145] W. Huggins, P. Patil, B. Mitchell, K. B. Whaley, and E. M. Stoudenmire, “Towards quantum machine learning with tensor networks,” *Quantum Science and technology*, vol. 4, no. 2, p. 024001, 2019.
- [146] Z. Xu, F. Yan, and Y. Qi, “Infinite Tucker decomposition: nonparametric bayesian models for multiway data analysis,” in *ICML*, 2012.
- [147] S. Zhe, Y. Qi, Y. Park, Z. Xu, I. Molloy, and S. Chari, “Dintucker: Scaling up gaussian process models on large multidimensional arrays,” in *AAAI*, 2016.
- [148] R. Penrose, “Applications of negative dimensional tensors,” *Combinatorial mathematics and its applications*, vol. 1, pp. 221–244, 1971.
- [149] Z. Xie, H. Liao, R. Huang, H. Xie, J. Chen, Z. Liu, and T. Xiang, “Optimized contraction scheme for tensor-network states,” *Physical Review B*, vol. 96, no. 4, p. 045128, 2017.
- [150] T. G. Kolda and B. W. Bader, “Tensor decompositions and applications,” *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.
- [151] 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 Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.
- [152] U. Schollwöck, “Matrix product state algorithms: DMRG, TEBD and relatives,” in *Strongly Correlated Systems*. Springer, 2013, pp. 67–98.
- [153] C. J. Hillar and L.-H. Lim, “Most tensor problems are NP-hard,” *Journal of the ACM*, vol. 60, no. 6, pp. 1–39, 2013.
- [154] L. De Lathauwer, B. De Moor, and J. Vandewalle, “A multilinear singular value decomposition,” *SIAM journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.
- [155] U. Schollwöck, “The density-matrix renormalization group in the age of matrix product states,” *Annals of Physics*, vol. 326, no. 1, pp. 96–192, 2011.
- [156] B. Pirvu, V. Murg, J. I. Cirac, and F. Verstraete, “Matrix product operator representations,” *New Journal of Physics*, vol. 12, no. 2, p. 025012, 2010.
- [157] H. Huang, Y. Liu, and C. Zhu, “Low-rank tensor grid for image completion,” *arXiv preprint arXiv:1903.04735*, 2019.
- [158] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *ECCV*, 2014.
- [159] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, “Importance estimation for neural network pruning,” in *CVPR*, 2019, pp. 11 264–11 272.
- [160] E. D. Karnin, “A simple procedure for pruning back-propagation trained neural networks,” *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 239–242, 1990.
- [161] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X. Hua, “Quantization networks,” in *CVPR*, 2019.
- [162] Y. Zhou, S. Moosavi-Dezfooli, N. Cheung, and P. Frossard, “Adaptive quantization for deep neural network,” in *AAAI*, 2018.
- [163] G. Hinton, O. Vinyals, J. Dean *et al.*, “Distilling the knowledge in a neural network,” *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.
- [164] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, “Tinybert: Distilling BERT for natural language understanding,” in *EMNLP*, 2020.
- [165] S. Chen, J. Zhou, W. Sun, and L. Huang, “Joint matrix decomposition for deep convolutional neural networks compression,” *arXiv preprint arXiv:2107.04386*, 2021.
- [166] T. N. Sainath, B. Kingsbury, V. Sindhvani, E. Arisoy, and B. Ramabhadran, “Low-rank matrix factorization for deep neural network training with high-dimensional output targets,” in *ICASSP*, 2013.
- [167] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [168] C. Lubich, T. Rohwedder, R. Schneider, and B. Vandereycken, “Dynamical approximation by hierarchical Tucker and tensor-train tensors,” *SIAM Journal on Matrix Analysis and Applications*, vol. 34, no. 2, pp. 470–494, 2013.
- [169] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” *AI Open*, vol. 1, pp. 57–81, 2020.
- [170] C. Chen, K. Batselier, C.-Y. Ko, and N. Wong, “Matrix product operator restricted Boltzmann machines,” in *IJCNN*, 2019.
- [171] L. Morency, R. Mihalcea, and P. Doshi, “Towards multimodal sentiment analysis: harvesting opinions from the web,” in *ICMI*, 2011.
- [172] H. Wang, A. Meghawat, L. Morency, and E. P. Xing, “Select-additive learning: Improving cross-individual generalization in multimodal sentiment analysis,” *CoRR*, vol. abs/1609.05244, 2016.
- [173] G. G. Chrysos, S. Moschoglou, G. Bouritsas, J. Deng, Y. Panagakis, and S. Zafeiriou, “Deep polynomial neural networks,” *IEEE Trans. PAMI*, vol. 44, no. 8, pp. 4021–4034, 2021.
- [174] J. D. Hidary and J. D. Hidary, *Quantum computing: an applied approach*. Springer, 2021, vol. 1.
- [175] N. Zettili, “Quantum mechanics: concepts and applications,” 2003.
- [176] H. Weimer, M. Müller, I. Lesanovsky, P. Zoller, and H. P. Büchler, “A Rydberg quantum simulator,” *Nature Physics*, vol. 6, no. 5, pp. 382–388, 2010.
- [177] J. Eisert, M. Friesdorf, and C. Gogolin, “Quantum many-body systems out of equilibrium,” *Nature Physics*, vol. 11, no. 2, pp. 124–130, 2015.
- [178] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *FOCS*, 1994.
- [179] A. Steane, “Quantum computing,” *Reports on Progress in Physics*, vol. 61, no. 2, p. 117, 1998.
- [180] N. Cohen and A. Shashua, “Convolutional rectifier networks as generalized tensor decompositions,” in *ICML*, 2016.
- [181] D. P. DiVincenzo, “Quantum computation,” *Science*, vol. 270, no. 5234, pp. 255–261, 1995.
- [182] G. Gan, P. Zhang, S. Li, X. Lu, and B. Wang, “Morphte: Injecting morphology in tensorized embeddings,” in *NeurIPS*, 2022.
- [183] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, “Parameterized quantum circuits as machine learning models,” *Quantum Science and Technology*, vol. 4, no. 4, p. 043001, 2019.
- [184] C. Hubig, I. P. McCulloch, U. Schollwöck, and F. A. Wolf, “Strictly single-site DMRG algorithm with subspace expansion,” *Physical Review B*, vol. 91, no. 15, p. 155115, 2015.
- [185] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, “Unsupervised generative modeling using matrix product states,” *Physical Review X*, vol. 8, no. 3, p. 031012, 2018.
- [186] S. Cheng, L. Wang, and P. Zhang, “Supervised learning with projected entangled pair states,” *Physical Review B*, vol. 103, no. 12, p. 125117, 2021.
- [187] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.
- [188] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [189] M. Born, “Quantenmechanik der stoßvorgänge,” *Zeitschrift für physik*, vol. 38, no. 11, pp. 803–827, 1926.
- [190] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, “A tutorial on energy-based learning,” *Predicting structured data*, vol. 1, no. 0, 2006.
- [191] J. P. Garrahan, “Classical stochastic dynamics and continuous matrix product states: gauge transformations, conditioned and driven processes, and equivalence of trajectory ensembles,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2016, no. 7, p. 073208, 2016.
- [192] Y. Levine, O. Sharir, N. Cohen, and A. Shashua, “Quantum entanglement in deep learning architectures,” *Physical review letters*, vol. 122, no. 6, p. 065301, 2019.
- [193] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *CVPR*, 2016.

- [194] C. Li and Z. Sun, “Evolutionary topology search for tensor network decomposition,” in *ICML*, 2020.
- [195] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *AISTATS*, 2010.
- [196] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on image classification,” in *ICCV*, 2015.
- [197] T. Elsken, J. H. Metzen, and F. Hutter, “Neural architecture search: A survey,” *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.
- [198] S. Nakajima, R. Tomioka, M. Sugiyama, and S. D. Babacan, “Perfect dimensionality recovery by variational bayesian PCA,” in *NeurIPS*, 2012.
- [199] L. Huang, C. Deng, S. Ibrahim, X. Fu, and B. Yuan, “VLSI hardware architecture of stochastic low-rank tensor decomposition,” in *ACSCC*, 2021.
- [200] N. Srivastava, H. Rong, P. Barua, G. Feng, H. Cao, Z. Zhang, D. Albonesi, V. Sarkar, W. Chen, P. Petersen *et al.*, “T2S-Tensor: Productively generating high-performance spatial hardware for dense tensor computations,” in *FCCM*, 2019.
- [201] N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, “Tensaurus: A versatile accelerator for mixed sparse-dense tensor computations,” in *HPCA*, 2020.
- [202] L. Liang, J. Xu, L. Deng, M. Yan, X. Hu, Z. Zhang, G. Li, and Y. Xie, “Fast search of the optimal contraction sequence in tensor networks,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 3, pp. 574–586, 2021.
- [203] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatin, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, “Discovering faster matrix multiplication algorithms with reinforcement learning,” *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.
- [204] V. Pereyra and G. Scherer, “Efficient computer manipulation of tensor products with applications to multidimensional approximation,” *Mathematics of Computation*, vol. 27, no. 123, pp. 595–605, 1973.
- [205] S. van der Walt, S. C. Colbert, and G. Varoquaux, “The NumPy array: A structure for efficient numerical computation,” *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, 2011.
- [206] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “Tensorflow: A system for large-scale machine learning,” in *USENIX*, K. Keeton and T. Roscoe, Eds., 2016.
- [207] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, “Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems,” *arXiv preprint arXiv:1512.01274*, 2015, 2015.
- [208] X. Liu, “TenDeC++: Tensor decomposition library in c++,” 2020. [Online]. Available: <https://github.com/osmint/TenDeC>
- [209] Z. Cai and J. Liu, “Approximating quantum many-body wave functions using artificial neural networks,” *Physical Review B*, vol. 97, no. 3, p. 035116, 2018.
- [210] K. Choo, G. Carleo, N. Regnault, and T. Neupert, “Symmetries and many-body excitations with neural-network quantum states,” *Physical review letters*, vol. 121, no. 16, p. 167204, 2018.
- [211] H. He, Y. Zheng, B. A. Bernevig, and N. Regnault, “Entanglement entropy from tensor network states for stabilizer codes,” *Physical Review B*, vol. 97, no. 12, p. 125102, 2018.