Tensor Networks for Dimensionality Reduction, Big Data and Deep Learning

Andrzej Cichocki

Abstract Large scale multidimensional data are often available as multiway arrays or higher-order tensors which can be approximately represented in distributed forms via low-rank tensor decompositions and tensor networks. Our particular emphasis is on elucidating that, by virtue of the underlying low-rank approximations, tensor networks have the ability to reduce the dimensionality and alleviate the curse of dimensionality in a number of applied areas, especially in large scale optimization problems and deep learning. We briefly review and provide tensor links between low-rank tensor network decompositions and deep neural networks. We elucidating, through graphical illustrations, that low-rank tensor approximations and sophisticated contractions of core tensors, tensor networks have the ability to perform distributed computations on otherwise prohibitively large volume of data/parameters. Our focus is on the Hierarchical Tucker, tensor train (TT) decompositions and MERA tensor networks in specific applications.

1 Introduction and Objectives

This paper aims to present some new ideas and methodologies related to tensor decompositions (TDs) and tensor networks models (TNs), especially in applications to deep neural networks (DNNs) and dimensionality reduction. The resurgence of artificial neural systems, especially deep learning neural networks has formed an active frontier of machine learning, signal processing and data mining [1–6, 13, 14]. Tensor decompositions (TDs) decompose complex data tensors of exceedingly high volume into their factor (component) matrices, while tensor networks (TNs)

A. Cichocki (✉)
Systems Research Institute, Polish Academy of Science, Warsaw, Poland
e-mail: a.cichocki@riken.jp

A. Cichocki
RIKEN Brain Science Institute, Tokyo, Japan

A. Cichocki
SKOLTECH, Moscow, Russia

A.E. Gawęda et al. (eds.), *Advances in Data Analysis with Computational Intelligence Methods*, Studies in Computational Intelligence 738,
https://doi.org/10.1007/978-3-319-67946-4_1

decompose higher-order tensors into sparsely interconnected small-scale factor matrices and/or low-order core tensors [7–14]. These low-order core tensors are called "components", "blocks", "factors" or simply "cores". In this way, large-scale data can be approximately represented in highly compressed and distributed formats.

In this paper, the TDs and TNs are treated in a unified way, by considering TDs as simple tensor networks or sub-networks; the terms "tensor decompositions" and "tensor networks" will therefore be used interchangeably. Tensor networks can be thought of as special graph structures which break down high-order tensors into a set of sparsely interconnected low-order core tensors, thus allowing for both enhanced interpretation and computational advantages [12–14].

Tensor networks offer a theoretical and computational framework for the analysis of computationally prohibitive large volumes of data, by "dissecting" such data into the "relevant" and "irrelevant" information. In this way, tensor network representations often allow for super-compression of data sets as large as 10^8 entries, down to the affordable levels of 10^5 or even less entries [15–25].

Challenges in Big Data Processing. Extreme-scale volumes and variety of modern data are becoming ubiquitous across the science and engineering disciplines. In the case of multimedia (speech, video), remote sensing and medical/biological data, the analysis also requires a paradigm shift in order to efficiently process massive data sets within tolerable time (velocity). Such massive data sets may have billions of entries and are typically represented in the form of huge block matrices and/or tensors. This has spurred a renewed interest in the development of tensor algorithms that are suitable for extremely large-scale data sets.

Apart from the huge Volume, the other features which characterize big data include Veracity, Variety and Velocity (see Fig. 1a and b). Each of the "V features" represents a research challenge in its own right. For example, high volume implies the need for algorithms that are scalable; high Velocity requires the processing of big data streams in near real-time; high Veracity calls for robust and predictive algorithms for noisy, incomplete and/or inconsistent data; high Variety demands the fusion of different data types, e.g., continuous, discrete, binary, time series, images, video, text, probabilistic or multi-view. Some applications give rise to additional "V challenges", such as Visualization, Variability and Value. The Value feature is particularly interesting and refers to the extraction of high quality and consistent information, from which meaningful and interpretable results can be obtained.

Our objective is to show that tensor networks provide a natural sparse and distributed representation for big data, and address both established and emerging methodologies for tensor-based representations and optimization. Our particular focus is on low-rank tensor network representations, which allow for huge data tensors to be approximated (compressed) by interconnected low-order core tensors [10, 11, 14].

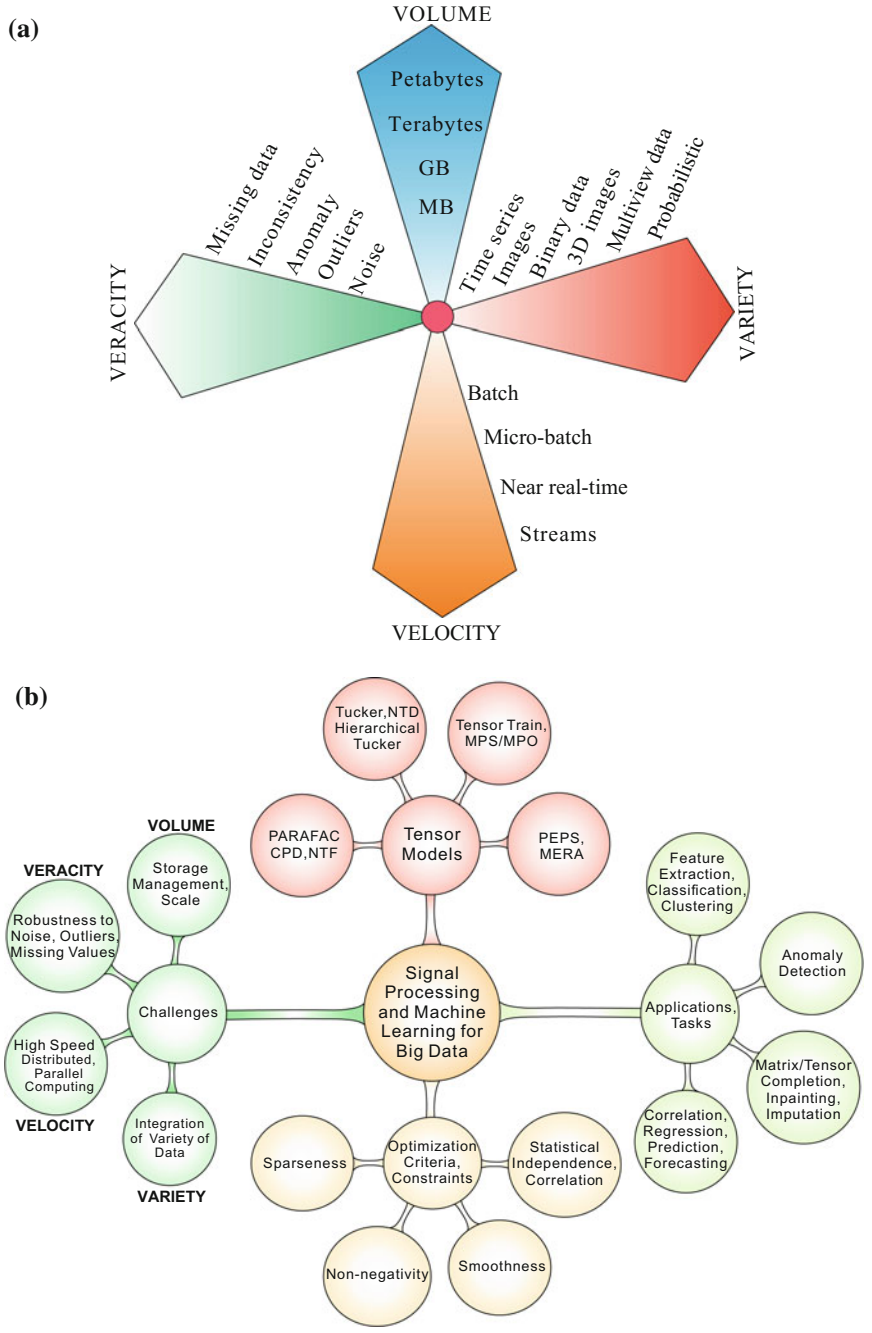

Fig. 1 **a** The 4 V challenges for big data. **b** A framework for extremely large-scale data analysis and the potential applications based on tensor decomposition approaches

2 Tensor Operations and Graphical Representations of Tensor Networks

Tensors are multi-dimensional generalizations of matrices. A matrix (2nd-order tensor) has two modes, rows and columns, while an Nth-order tensor has N modes for example, a 3rd-order tensor (with three-modes) looks like a cube. Sub-tensors are formed when a subset of tensor indices is fixed. Of particular interest are fibers which are vectors obtained by fixing every tensor index but one, and matrix slices which are two-dimensional sections (matrices) of a tensor, obtained by fixing all the tensor indices but two. It should be noted that block matrices can also be represented by tensors.

We adopt the notation whereby tensors (for $N \geq 3$) are denoted by bold underlined capital letters, e.g., $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. For simplicity, we assume that all tensors are real-valued, but it is possible to define tensors as complex-valued or over arbitrary fields. Matrices are denoted by boldface capital letters, e.g., $\mathbf{X} \in \mathbb{R}^{I \times J}$, and vectors (1st-order tensors) by boldface lower case letters, e.g., $\mathbf{x} \in \mathbb{R}^J$. For example, the columns of the matrix $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_R] \in \mathbb{R}^{I \times R}$ are the vectors denoted by $\mathbf{a}_r \in \mathbb{R}^I$, while the elements of a matrix (scalars) are denoted by lowercase letters, e.g., $a_{ir} = \mathbf{A}(i, r)$ (for more details regarding notations and basic tensor operations see [10–14, 26].

A specific entry of an Nth-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is denoted by $x_{i_1, i_2, \ldots, i_N} = \underline{\mathbf{X}}(i_1, i_2, \ldots, i_N) \in \mathbb{R}$. The order of a tensor is the number of its "modes", "ways" or "dimensions", which can include space, time, frequency, trials, classes, and dictionaries. The term "size" stands for the number of values that an index can take in a particular mode. For example, the tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is of order N and size I_n in all modes-n ($n = 1, 2, \ldots, N$). Lower-case letters e.g., i, j are used for the subscripts in running indices and capital letters I, J denote the upper bound of an index, i.e., $i = 1, 2, \ldots, I$ and $j = 1, 2, \ldots, J$. For a positive integer n, the shorthand notation $<n>$ denotes the set of indices $\{1, 2, \ldots, n\}$.

Notations and terminology used for tensors and tensor networks differ across the scientific communities to this end we employ a unifying notation particularly suitable for machine learning and signal processing research [13, 14].

A precise description of tensors and tensor operations is often tedious and cumbersome, given the multitude of indices involved. We grossly simplify the description of tensors and their mathematical operations through diagrammatic representations borrowed from physics and quantum chemistry (see [13, 14, 27] and references therein). In this way, tensors are represented graphically by nodes of any geometrical shapes (e.g., circles, squares, dots), while each outgoing line ("edge", "leg", "arm") from a node represents the indices of a specific mode (see Fig. 2a). In our adopted notation, each scalar (zero-order tensor), vector (first-order tensor), matrix (2nd-order tensor), 3rd-order tensor or higher-order tensor is represented by a circle (or rectangular), while the order of a tensor is determined by the number of lines (edges) connected to it. According to this notation, an Nth-order tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is represented by a circle (or any shape) with N branches each of size I_n, $n = 1, 2, \ldots, N$

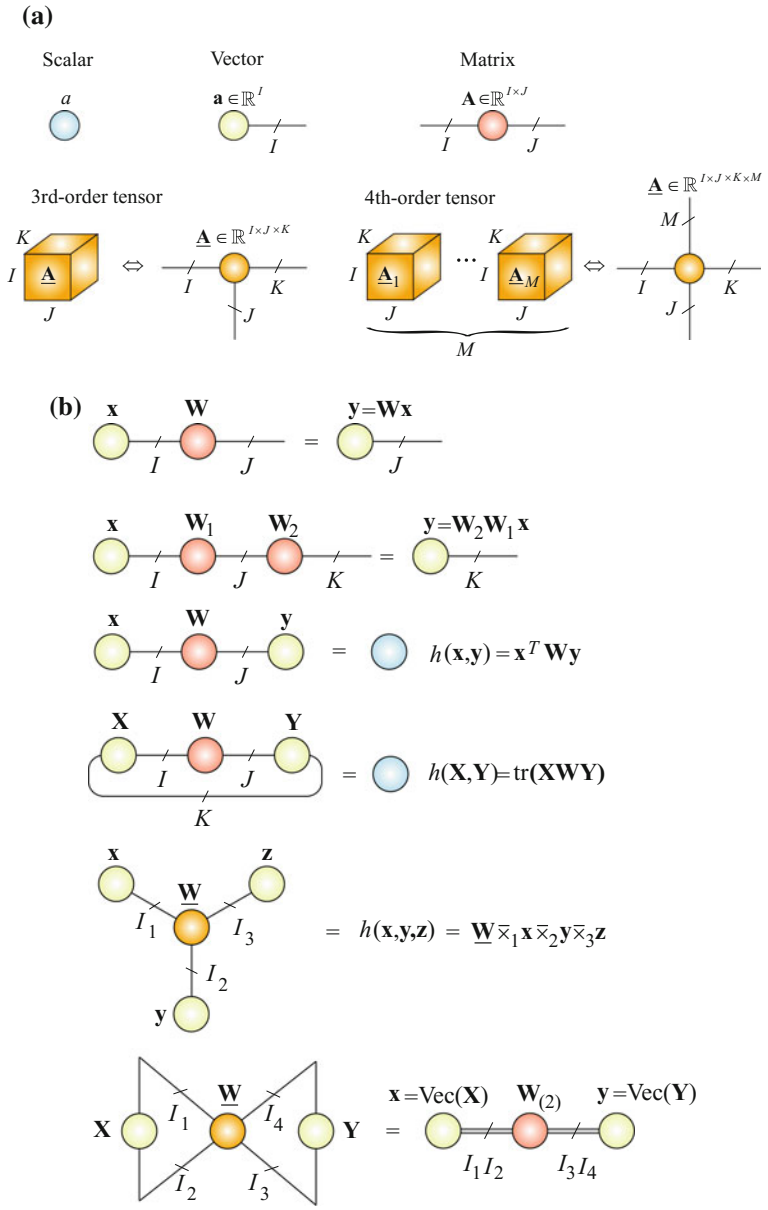

Fig. 2 Graphical representation of tensor operations. **a** Basic building blocks for tensor network diagrams. **b** Tensor network diagrams for matrix-vector and tensor-vectors multiplications

(see Sect. 2.1). An interconnection between two circles designates a contraction of tensors, which is a summation of products over a common index (see Fig. 2b).

Hierarchical (multilevel block) matrices are also naturally represented by tensors. All mathematical operations on tensors can be therefore equally performed on block matrices [12, 13].

In this paper, we make extensive use of tensor network diagrams as an intuitive and visual way to efficiently represent tensor decompositions. Such graphical notations are of great help in studying and implementing sophisticated tensor operations. We highlight the significant advantages of such diagrammatic notations in the description of tensor manipulations, and show that most tensor operations can be visualized through changes in the architecture of a tensor network diagram.

2.1 Tensor Operations and Tensor Network Diagrams

Tensor operations benefit from the power of multilinear algebra which is structurally much richer than linear algebra, and even some basic properties, such as the rank, have a more complex meaning.

For convenience, general operations, such as vec(·) or diag(·), are defined similarly to the MATLAB syntax.

Multi-indices: By a multi-index $i = \overline{i_1 i_2 \cdots i_N}$ we refer to an index which takes all possible combinations of values of indices, $i_1, i_2, \ldots, i_N$, for $i_n = 1, 2, \ldots, I_n$, $n = 1, 2, \ldots, N$ and in a specific order. Multi–indices can be defined using the following convention [28]:

$$\overline{i_1 i_2 \cdots i_N} = i_N + (i_{N-1} - 1)I_N + (i_{N-2} - 1)I_N I_{N-1} + \cdots + (i_1 - 1)I_2 \cdots I_N.$$

Matricization. The matricization operator, also known as the unfolding or flattening, reorders the elements of a tensor into a matrix. Such a matrix is re-indexed according to the choice of multi-index described above, and the following two fundamental matricizations are used extensively.

The mode-n matricization. For a fixed index $n \in \{1, 2, \ldots, N\}$, the mode-$n$ matricization of an Nth-order tensor, $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, is defined as the ("short" and "wide") matrix

$$\mathbf{X}_{(n)} \in \mathbb{R}^{I_n \times I_1 I_2 \cdots I_{n-1} I_{n+1} \cdots I_N}, \tag{1}$$

with I_n rows and $I_1 I_2 \cdots I_{n-1} I_{n+1} \cdots I_N$ columns, the entries of which are

$$(\mathbf{X}_{(n)})_{i_n, \overline{i_1 \ldots i_{n-1} i_{n+1} \ldots i_N}} = x_{i_1, i_2, \ldots, i_N}.$$

Note that the columns of a mode-n matricization, $\mathbf{X}_{(n)}$, of a tensor $\underline{\mathbf{X}}$ are the mode-n fibers of $\underline{\mathbf{X}}$.

The mode-$\{n\}$ canonical matricization. For a fixed index $n \in \{1, 2, \dots, N\}$, the mode-$(1, 2, \dots, n)$ matricization, or simply mode-n canonical matricization, of a tensor $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is defined as the matrix

$$\mathbf{X}_{<n>} \in \mathbb{R}^{I_1 I_2 \cdots I_n \times I_{n+1} \cdots I_N}, \tag{2}$$

with $I_1 I_2 \cdots I_n$ rows and $I_{n+1} \cdots I_N$ columns, and the entries

$$(\mathbf{X}_{<n>})_{\overline{i_1 i_2 \dots i_n}, \overline{i_{n+1} \dots i_N}} = x_{i_1, i_2, \dots, i_N}.$$

The matricization operator in the MATLAB notation (reverse lexicographic) is given by

$$\mathbf{X}_{<n>} = \text{reshape}\left(\underline{\mathbf{X}}, I_1 I_2 \cdots I_n, I_{n+1} \cdots I_N\right). \tag{3}$$

As special cases we immediately have

$$\mathbf{X}_{<1>} = \mathbf{X}_{(1)}, \quad \mathbf{X}_{<N-1>} = \mathbf{X}_{(N)}^{\mathrm{T}}, \quad \mathbf{X}_{<N>} = \text{vec}(\mathbf{X}). \tag{4}$$

The tensorization of a vector or a matrix can be considered as a reverse process to the vectorization or matricization (see Fig. 3) [14].

The following symbols are used for most common tensor multiplications: $\circ$ for the outer product $\otimes$ for the Kronecker product, $\odot$ for the Khatri–Rao product, $\circledast$ for the Hadamard (componentwise) product, and $\times_n$ for the mode-n product. We refer to [13, 14, 26, 29] for more detail regarding the basic notations and tensor operations (Figs. 4 and 5).

Outer product. The central operator in tensor analysis is the outer or tensor product, which for the tensors $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\underline{\mathbf{B}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$ gives the tensor $\underline{\mathbf{C}} = \underline{\mathbf{A}} \circ \underline{\mathbf{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_N \times J_1 \times \cdots \times J_M}$ with entries $c_{i_1, \dots, i_N, j_1, \dots, j_M} = a_{i_1, \dots, i_N}\, b_{j_1, \dots, j_M}$.

Note that for 1st-order tensors (vectors), the tensor product reduces to the standard outer product of two nonzero vectors, $\mathbf{a} \in \mathbb{R}^I$ and $\mathbf{b} \in \mathbb{R}^J$, which yields a rank-1 matrix, $\mathbf{X} = \mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^{\mathrm{T}} \in \mathbb{R}^{I \times J}$. The outer product of three nonzero vectors, $\mathbf{a} \in \mathbb{R}^I$, $\mathbf{b} \in \mathbb{R}^J$ and $\mathbf{c} \in \mathbb{R}^K$, gives a 3rd-order rank-1 tensor (called pure or elementary tensor), $\underline{\mathbf{X}} = \mathbf{a} \circ \mathbf{b} \circ \mathbf{c} \in \mathbb{R}^{I \times J \times K}$, with entries $x_{ijk} = a_i\, b_j\, c_k$.

The outer (tensor) product has been generalized to the nonlinear outer (tensor) products, as follows

$$\left(\underline{\mathbf{A}} \circ_{\rho} \underline{\mathbf{B}}\right)_{i_1, \dots, i_N, j_1, \dots, j_M} = \rho\left(a_{i_1, \dots, i_N}, b_{j_1, \dots, j_M}\right), \tag{5}$$

where ρ is, in general, nonlinear suitably chosen function (see [30] and Sect. 7 for more detail).

In a similar way, we can define the generalized Kronecker and the Khatri-Rao products. Generalized Kronecker product of two tensors $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and

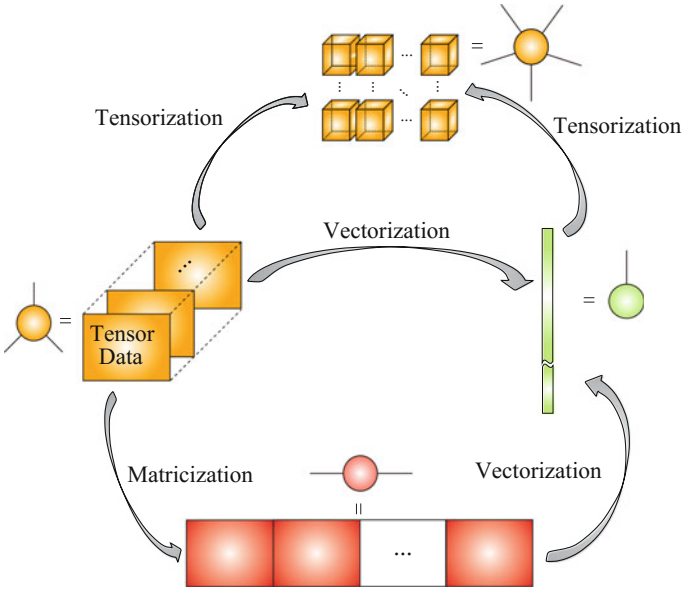

Fig. 3 Tensor reshaping operations: matricization, vectorization and tensorization. Matricization refers to converting a tensor into a matrix, vectorization to converting a tensor or a matrix into a vector, while tensorization refers to converting a vector, a matrix or a low-order tensor into a higher-order tensor

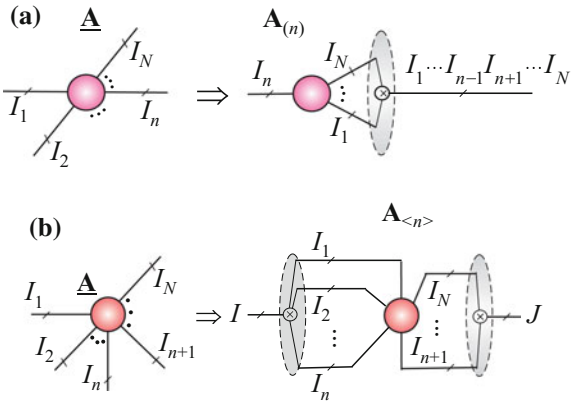

Fig. 4 Matricization (flattening, unfolding) used in tensor reshaping. **a** Tensor network diagram for the mode-n matricization of an Nth-order tensor, $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, into a short and wide matrix, $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times I_1 \cdots I_{n-1} I_{n+1} \cdots I_N}$. **b** Mode-$\{1, 2, \ldots, n\}$th (canonical) matricization of an Nth-order tensor, $\underline{\mathbf{A}}$, into a matrix $\mathbf{A}_{<n>} = \mathbf{A}_{(\overline{i_1 \ldots i_n}\,;\, \overline{i_{n+1} \ldots i_N})} \in \mathbb{R}^{I_1 I_2 \cdots I_n \times I_{n+1} \cdots I_N}$

$\underline{\mathbf{B}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_N}$ yields a tensor $\underline{\mathbf{C}} = \underline{\mathbf{A}} \otimes_\rho \underline{\mathbf{B}} \in \mathbb{R}^{I_1 J_1 \times \cdots \times I_N J_N}$, with entries $c_{\overline{i_1 j_1}, \ldots, \overline{i_N j_N}} = \rho(a_{i_1,\ldots,i_N}, b_{j_1,\ldots,j_N})$.

Analogously, we can define a generalized Khatri–Rao product of two matrices $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_J] \in \mathbb{R}^{I \times J}$ and $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_J] \in \mathbb{R}^{K \times J}$ is a matrix $\mathbf{C} = \mathbf{A} \odot_\rho \mathbf{B} \in \mathbb{R}^{IK \times J}$, with columns $\mathbf{c}_j = \mathbf{a}_j \otimes_\rho \mathbf{b}_j \in \mathbb{R}^{IK}$.

CP decomposition, Kruskal tensor. Any tensor can be expressed as a finite sum of rank-1 tensors, in the form

$$\underline{\mathbf{X}} = \sum_{r=1}^{R} \mathbf{b}_r^{(1)} \circ \mathbf{b}_r^{(2)} \circ \cdots \circ \mathbf{b}_r^{(N)} = \sum_{r=1}^{R} \left(\bigcirc_{n=1}^{N} \mathbf{b}_r^{(n)} \right), \quad \mathbf{b}_r^{(n)} \in \mathbb{R}^{I_n}, \tag{6}$$

which is exactly the form of the Kruskal tensor, also known under the names of CANDECOMP/PARAFAC, Canonical Polyadic Decomposition (CPD), or simply the CP decomposition in (23). We will use the acronyms CP and CPD.

Tensor rank. The tensor rank, also called the CP rank, is a natural extension of the matrix rank and is defined as a minimum number, R, of rank-1 terms in an exact CP decomposition of the form in (6).

Multilinear products. The mode-n (multilinear) product, also called the tensor-times-matrix product (TTM), of a tensor, $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$, and a matrix, $\mathbf{B} \in \mathbb{R}^{J \times I_n}$, gives the tensor

$$\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_n \mathbf{B} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}, \tag{7}$$

with entries $c_{i_1,i_2,\ldots,i_{n-1},j,i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{I_n} a_{i_1,i_2,\ldots,i_N}\, b_{j,i_n}$. An equivalent matrix representation is $\mathbf{C}_{(n)} = \mathbf{B}\mathbf{A}_{(n)}$, which allows us to employ established fast matrix-by-vector and matrix-by-matrix multiplications when dealing with very large-scale tensors. Efficient and optimized algorithms for TTM are, however, still emerging [31–33].

Full Multilinear Product. A full multilinear product, also called the Tucker product,[1] of an Nth-order tensor, $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$, and a set of N factor matrices, $\underline{\mathbf{B}}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n = 1, 2, \ldots, N$, performs the multiplications in all the modes and can be compactly written as

$$\underline{\mathbf{C}} = \underline{\mathbf{G}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \cdots \times_N \mathbf{B}^{(N)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}.$$

Observe that this format corresponds to the Tucker decomposition [26, 34, 35] (see also Sect. 3.1).

Multilinear product of a tensor and a vector (TTV). In a similar way, the mode-n multiplication of a tensor, $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$, and a vector, $\mathbf{b} \in \mathbb{R}^{R_n}$ (tensor-times-vector, TTV) yields a tensor

[1] The standard multilinear product can be generalized to nonlinear multilinear product as $\underline{\mathbf{C}} = \underline{\mathbf{G}} \times_1^\sigma \mathbf{B}^{(1)} \times_2^\sigma \mathbf{B}^{(2)} \cdots \times_N^\sigma \mathbf{B}^{(N)}$, where $\underline{\mathbf{G}} \times_n^\sigma \mathbf{B} = \sigma(\underline{\mathbf{G}} \times_n \mathbf{B})$, and σ is a suitably chosen nonlinear activation function.

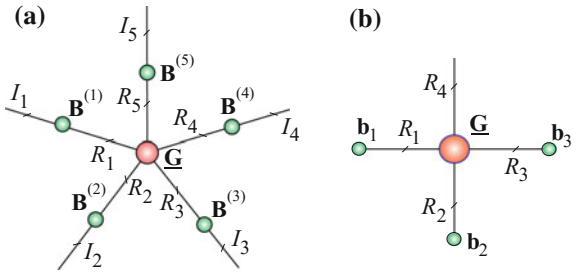

Fig. 5 Generalized (nonlinear) multilinear tensor products used in deep learning in a compact tensor network notation. **a** Generalized multilinear product of a tensor, $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_5}$, and five factor (component) matrices, $\mathbf{B}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ $(n = 1, 2, \ldots, 5)$, yields the tensor $\underline{\mathbf{C}} = (((((\underline{\mathbf{G}} \times_1^{\sigma} \mathbf{B}^{(1)}) \times_2^{\sigma} \mathbf{B}^{(2)}) \times_3^{\sigma} \mathbf{B}^{(3)}) \times_4^{\sigma} \mathbf{B}^{(4)}) \times_5^{\sigma} \mathbf{B}^{(5)}) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_5}$. This corresponds to the generalized Tucker format. **c** Generalized multi-linear product of a 4th-order tensor, $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times R_2 \times R_3 \times R_4}$, and three vectors, $\mathbf{b}_n \in \mathbb{R}^{R_n}$ $(n = 1, 2, 3)$, yields the vector $\mathbf{c} = (((\underline{\mathbf{G}} \bar{\times}_1^{\sigma} \mathbf{b}_1) \bar{\times}_2^{\sigma} \mathbf{b}_2) \bar{\times}_3^{\sigma} \mathbf{b}_3) \in \mathbb{R}^{R_4}$, where, in general, σ is a nonlinear activation function

$$\underline{\mathbf{C}} = \underline{\mathbf{G}} \bar{\times}_n \mathbf{b} \in \mathbb{R}^{R_1 \times \cdots \times R_{n-1} \times R_{n+1} \times \cdots \times R_N}, \tag{8}$$

with entries $c_{r_1,\ldots,r_{n-1},r_{n+1},\ldots,r_N} = \sum_{r_n=1}^{R_n} g_{r_1,\ldots,r_{n-1},r_n,r_{n+1},\ldots,r_N} \, b_{r_n}$.

Note that the mode-n multiplication of a tensor by a matrix does not change the tensor order, while the multiplication of a tensor by vectors reduces its order, with the mode n removed.

Multilinear products of tensors by matrices or vectors play a key role in deterministic methods for the reshaping of tensors and dimensionality reduction, as well as in probabilistic methods for randomization/sketching procedures and in random projections of tensors into matrices or vectors. In other words, we can also perform reshaping of a tensor through random projections that change its entries, dimensionality or size of modes, and/or the tensor order. This is achieved by multiplying a tensor by random matrices or vectors, transformations which preserve its basic properties [36–43].

Tensor contractions. Tensor contraction is a fundamental and the most important operation in tensor networks, and can be considered a higher-dimensional analogue of matrix multiplication, inner product, and outer product.

In a way similar to the mode-n multilinear product,[2] the mode-$\binom{m}{n}$ product (tensor contraction) of two tensors, $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\underline{\mathbf{B}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$, with common modes, $I_n = J_m$, yields an $(N + M - 2)$-order tensor, $\underline{\mathbf{C}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N \times J_1 \times \cdots \times J_{m-1} \times J_{m+1} \times \cdots \times J_M}$, in the form (see Fig. 6a)

$$\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_n^m \underline{\mathbf{B}}, \tag{9}$$

[2] In the literature, sometimes the symbol $\times_n$ is replaced by $\bullet_n$.

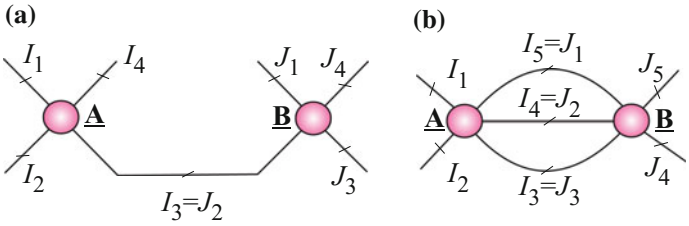

Fig. 6 Examples of contractions of two tensors. **a** Tensor contraction of two 4th-order tensors, along mode-3 in $\underline{\mathbf{A}}$ and mode-2 in $\underline{\mathbf{B}}$, yields a 6th-order tensor, $\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_3^2 \underline{\mathbf{B}} \in \mathbb{R}^{I_1 \times I_2 \times I_4 \times J_1 \times J_3 \times J_4}$, with entries $c_{i_1,i_2,i_4,j_1,j_3,j_4} = \sum_{i_3} a_{i_1,i_2,i_3,i_4}\, b_{j_1,i_3,j_3,j_4}$. **b** Tensor contraction of two 5th-order tensors along the modes 3, 4, 5 in $\underline{\mathbf{A}}$ and 1, 2, 3 in $\underline{\mathbf{B}}$ yields a 4th-order tensor, $\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_{5,4,3}^{1,2,3} \underline{\mathbf{B}} \in \mathbb{R}^{I_1 \times I_2 \times J_4 \times J_5}$. Nonlinear contraction can be also performed similar to formula (5)

for which the entries are computed as $c_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N,j_1,\ldots,j_{m-1},j_{m+1},\ldots,j_M} = \sum_{i_n=1}^{I_n} a_{i_1,\ldots,i_{n-1},i_n,i_{n+1},\ldots,i_N}\, b_{j_1,\ldots,j_{m-1},i_n,j_{m+1},\ldots,j_M}$. This operation is referred to as a contraction of two tensors in single common mode.

Tensors can be contracted in several modes (or even in all modes), as illustrated in Fig. 6. Often, the super- or sub-index, e.g., m, n, will be omitted in a few special cases. For example, the multilinear product of the tensors, $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\underline{\mathbf{B}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$, with common modes, $I_N = J_1$, can be written as

$$\underline{\mathbf{C}} = \underline{\mathbf{A}} \times_N^1 \underline{\mathbf{B}} = \underline{\mathbf{A}} \times^1 \underline{\mathbf{B}} = \underline{\mathbf{A}} \bullet \underline{\mathbf{B}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_{N-1} \times J_2 \times \cdots \times J_M}, \tag{10}$$

for which the entries $c_{\mathbf{i}_{2:N},\mathbf{j}_{2:M}} = \sum_{i=1}^{I_1} a_{i,\mathbf{i}_{2:N}} b_{i,\mathbf{j}_{2:M}}$ by using the MATLAB notation $\mathbf{i}_{p:q} = \{i_p, i_{p+1}, \ldots, i_{q-1}, i_q\}$.

In this notation, the multiplications of matrices and vectors can be written as, $\mathbf{A} \times_2^1 \mathbf{B} = \mathbf{A} \times^1 \mathbf{B} = \mathbf{AB}$, $\mathbf{A} \times_2^2 \mathbf{B} = \mathbf{AB}^{\mathrm{T}}$, $\mathbf{A} \times_{1,2}^{1,2} \mathbf{B} = \mathbf{A} \bar{\times} \mathbf{B} = \langle \mathbf{A}, \mathbf{B} \rangle$, and $\mathbf{A} \times_2^1 \mathbf{x} = \mathbf{A} \times^1 \mathbf{x} = \mathbf{Ax}$.

In practice, due to the high computational complexity of tensor contractions, especially for tensor networks with loops, this operation is often performed approximately [44–47].

3 Mathematical and Graphical Representation of Basic Tensor Networks

Tensor networks (TNs) represent a higher-order tensor as a set of sparsely interconnected lower-order tensors (see Fig. 7), and in this way provide computational and storage benefits. The lines (branches, edges) connecting core tensors correspond to the contracted modes while their weights (or numbers of branches) represent the

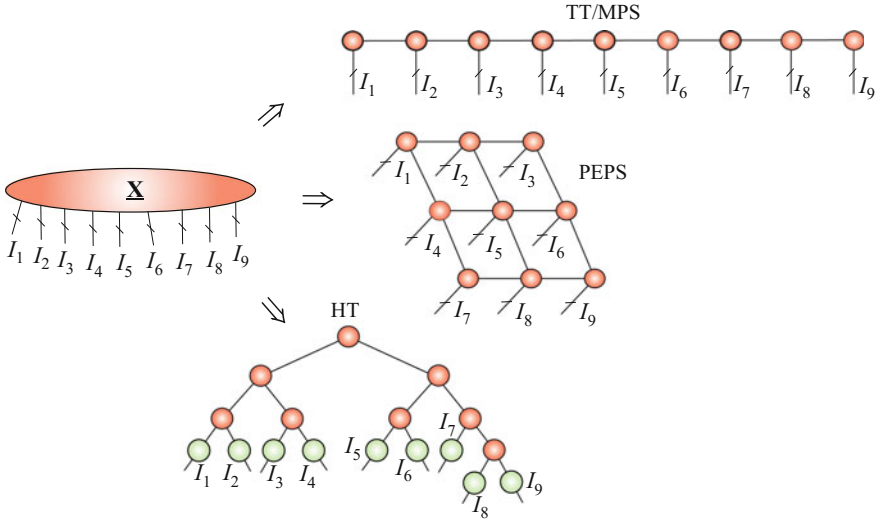

Fig. 7 Illustration of the decomposition of a 9th-order tensor, $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_9}$, into different forms of tensor networks (TNs). In general, the objective is to decompose a very high-order tensor into sparsely (weakly) connected low-order and small size core tensors, typically 3rd-order and 4th-order cores. Top: The Tensor Train (TT) model, which is equivalent to the Matrix Product State (MPS) with closed boundary conditions (CBC). Middle: The Projected Entangled-Pair States (PEPS). Bottom: The Hierarchical Tucker (HT)

rank of a tensor network,[3] whereas the lines which do not connect core tensors correspond to the "external" physical variables (modes, indices) within the data tensor. In other words, the number of free (dangling) edges (with weights larger than one) determines the order of a data tensor under consideration, while set of weights of internal branches represents the TN rank.

3.1 The CP and Tucker Tensor Formats

The CP and Tucker decompositions have long history. For recent surveys and more detailed information we refer to [12–14, 26, 48–50]. Compared to the CP decomposition, the Tucker decomposition provides a more general factorization of an Nth-order tensor into a relatively small size core tensor and factor matrices, and can be expressed as follows:

[3] Strictly speaking, the minimum set of internal indices $\{R_1, R_2, R_3, \ldots\}$ is called the rank (bond dimensions) of a specific tensor network.

$$\begin{aligned}\underline{\mathbf{X}} &\cong \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \ldots r_N} \left(\mathbf{b}_{r_1}^{(1)} \circ \mathbf{b}_{r_2}^{(2)} \circ \cdots \circ \mathbf{b}_{r_N}^{(N)} \right) \\ &= \underline{\mathbf{G}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \cdots \times_N \mathbf{B}^{(N)} \\ &= [\![\underline{\mathbf{G}}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \ldots, \mathbf{B}^{(N)}]\!], \end{aligned} \tag{11}$$

where $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is the given data tensor, $\underline{\mathbf{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ is the core tensor, and $\mathbf{B}^{(n)} = [\mathbf{b}_1^{(n)}, \mathbf{b}_2^{(n)}, \ldots, \mathbf{b}_{R_n}^{(n)}] \in \mathbb{R}^{I_n \times R_n}$ are the mode-n factor (component) matrices, $n = 1, 2, \ldots, N$ (see Fig. 8). The core tensor (typically, $R_n \ll I_n$) models a potentially complex pattern of mutual interaction between the vectors in different modes. The model in (11) is often referred to as the Tucker-N model.

Using the properties of the Kronecker tensor product, the Tucker-N decomposition in (11) can be expressed in an equivalent vector form as

$$\text{vec}(\underline{\mathbf{X}}) \cong [\mathbf{B}^{(N)} \otimes \mathbf{B}^{(N-1)} \otimes \cdots \otimes \mathbf{B}^{(1)}] \, \text{vec}(\underline{\mathbf{G}}), \tag{12}$$

where the multi-indices are ordered in a reverse lexicographic order (little-endian).

Note that the CP decomposition can be considered as a special case of the Tucker decomposition, whereby the cube core tensor has nonzero elements only on the main diagonal. In contrast to the CP decomposition, the unconstrained Tucker decomposition is not unique. However, constraints imposed on all factor matrices and/or core tensor can reduce the indeterminacies to only column-wise permutation and scaling, thus yielding a unique core tensor and factor matrices [51].

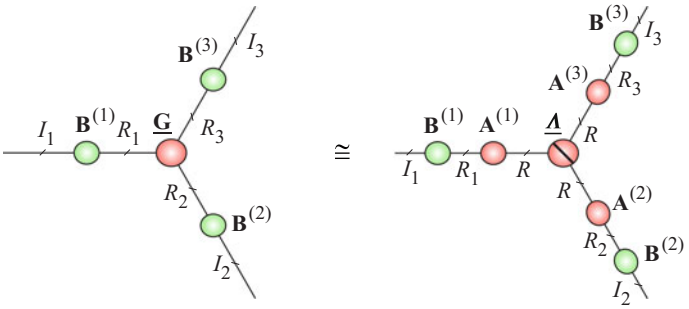

Fig. 8 Illustration of the standard Tucker and Tucker-CP decompositions, where the objective is to compute the factor matrices, $\mathbf{B}^{(n)}$, and the core tensor, $\underline{\mathbf{G}}$. Tucker decomposition of a 3rd-order tensor, $\underline{\mathbf{X}} \cong \underline{\mathbf{G}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \times_3 \mathbf{B}^{(3)}$. In some applications, the core tensor can be further approximately factorized using the CP decomposition as $\underline{\mathbf{G}} \cong \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r$ or alternatively using TT/HT decompositions. Graphical representation of the Tucker-CP decomposition for a 3rd-order tensor, $\underline{\mathbf{X}} \cong \underline{\mathbf{G}} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \times_3 \mathbf{B}^{(3)} = [\![\underline{\mathbf{G}}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \mathbf{B}^{(3)}]\!] \cong (\underline{\boldsymbol{\Lambda}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \times_3 \mathbf{A}^{(3)}) \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \times_3 \mathbf{B}^{(3)} = [\![\underline{\boldsymbol{\Lambda}}; \mathbf{B}^{(1)}\mathbf{A}^{(1)}, \mathbf{B}^{(2)}\mathbf{A}^{(2)}, \mathbf{B}^{(3)}\mathbf{A}^{(3)}]\!]$

3.2 Operations in the Tucker Format

If very large-scale data tensors admit an exact or approximate representation in their TN formats, then most mathematical operations can be performed more efficiently using the so obtained much smaller core tensors and factor matrices.

As illustrative example, consider the Nth-order tensors $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$ in the Tucker format, given by

$$\underline{\mathbf{X}} = [\![\underline{\mathbf{G}}_X; \mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}]\!] \quad \text{and} \quad \underline{\mathbf{Y}} = [\![\underline{\mathbf{G}}_Y; \mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(N)}]\!], \tag{13}$$

for which the respective multilinear ranks are $\{R_1, R_2, \ldots, R_N\}$ and $\{Q_1, Q_2, \ldots, Q_N\}$, then the following mathematical operations can be performed directly in the Tucker format, which admits a significant reduction in computational costs [13, 52–54]:

- **The addition** of two Tucker tensors of the same order and sizes

$$\underline{\mathbf{X}} + \underline{\mathbf{Y}} = [\![\underline{\mathbf{G}}_X \oplus \underline{\mathbf{G}}_Y;\ [\mathbf{X}^{(1)}, \mathbf{Y}^{(1)}], \ldots, [\mathbf{X}^{(N)}, \mathbf{Y}^{(N)}]]\!], \tag{14}$$

  where $\oplus$ denotes a direct sum of two tensors, and $[\mathbf{X}^{(n)}, \mathbf{Y}^{(n)}] \in \mathbb{R}^{I_n \times (R_n + Q_n)}$, $\mathbf{X}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ and $\mathbf{Y}^{(n)} \in \mathbb{R}^{I_n \times Q_n}$, $\forall n$.
- **The Kronecker product** of two Tucker tensors of arbitrary orders and sizes

$$\underline{\mathbf{X}} \otimes \underline{\mathbf{Y}} = [\![\underline{\mathbf{G}}_X \otimes \underline{\mathbf{G}}_Y;\ \mathbf{X}^{(1)} \otimes \mathbf{Y}^{(1)}, \ldots, \mathbf{X}^{(N)} \otimes \mathbf{Y}^{(N)}]\!]. \tag{15}$$

- **The Hadamard** or element-wise product of two Tucker tensors of the same order and the same sizes

$$\underline{\mathbf{X}} \circledast \underline{\mathbf{Y}} = [\![\underline{\mathbf{G}}_X \otimes \underline{\mathbf{G}}_Y;\ \mathbf{X}^{(1)} \odot_1 \mathbf{Y}^{(1)}, \ldots, \mathbf{X}^{(N)} \odot_1 \mathbf{Y}^{(N)}]\!], \tag{16}$$

  where $\odot_1$ denotes the mode-1 Khatri–Rao product, also called the transposed Khatri–Rao product or row-wise Kronecker product.
- **The inner product** of two Tucker tensors of the same order and sizes can be reduced to the inner product of two smaller tensors by exploiting the Kronecker product structure in the vectorized form, as follows

$$\begin{aligned} \langle \underline{\mathbf{X}}, \underline{\mathbf{Y}} \rangle &= \text{vec}(\underline{\mathbf{X}})^{\mathrm{T}}\, \text{vec}(\underline{\mathbf{Y}}) \\ &= \text{vec}(\underline{\mathbf{G}}_X)^{\mathrm{T}} \left(\bigotimes_{n=1}^{N} \mathbf{X}^{(n)\,\mathrm{T}} \right) \left(\bigotimes_{n=1}^{N} \mathbf{Y}^{(n)} \right) \text{vec}(\underline{\mathbf{G}}_Y) \\ &= \text{vec}(\underline{\mathbf{G}}_X)^{\mathrm{T}} \left(\bigotimes_{n=1}^{N} \mathbf{X}^{(n)\,\mathrm{T}}\, \mathbf{Y}^{(n)} \right) \text{vec}(\underline{\mathbf{G}}_Y) \\ &= \langle [\![\underline{\mathbf{G}}_X; (\mathbf{X}^{(1)\,\mathrm{T}}\, \mathbf{Y}^{(1)}), \ldots, (\mathbf{X}^{(N)\,\mathrm{T}}\, \mathbf{Y}^{(N)})]\!], \underline{\mathbf{G}}_Y \rangle. \end{aligned} \tag{17}$$

- **The Frobenius norm** can be computed in a particularly simple way if the factor matrices are orthogonal, since then all products $\mathbf{X}^{(n)\mathrm{T}}\mathbf{X}^{(n)}$, $\forall n$, become the identity matrices, so that

$$\begin{aligned}\|\mathbf{X}\|_F &= \langle \underline{\mathbf{X}}, \underline{\mathbf{X}} \rangle \\ &= \mathrm{vec}\left(\left[\!\left[\underline{\mathbf{G}}_X; (\mathbf{X}^{(1)\mathrm{T}}\mathbf{X}^{(1)}), \ldots, (\mathbf{X}^{(N)\mathrm{T}}\mathbf{X}^{(N)})\right]\!\right]\right)^{\mathrm{T}} \mathrm{vec}(\underline{\mathbf{G}}_X) \\ &= \mathrm{vec}(\underline{\mathbf{G}}_X)^{\mathrm{T}}\,\mathrm{vec}(\underline{\mathbf{G}}_X) = \|\underline{\mathbf{G}}_X\|_F. \end{aligned} \tag{18}$$

- **The N-D discrete convolution** of tensors $\underline{\mathbf{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\underline{\mathbf{Y}} \in \mathbb{R}^{J_1 \times \cdots \times J_N}$ in their Tucker formats can be expressed as

$$\begin{aligned}\underline{\mathbf{Z}} = \underline{\mathbf{X}} * \underline{\mathbf{Y}} &= \left[\!\left[\underline{\mathbf{G}}_Z; \mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(N)}\right]\!\right] \\ &\in \mathbb{R}^{(I_1+J_1-1)\times\cdots\times(I_N+J_N-1)}. \end{aligned} \tag{19}$$

  If $\{R_1, R_2, \ldots, R_N\}$ is the multilinear rank of $\underline{\mathbf{X}}$ and $\{Q_1, Q_2, \ldots, Q_N\}$ the multilinear rank $\underline{\mathbf{Y}}$, then the core tensor $\underline{\mathbf{G}}_Z = \underline{\mathbf{G}}_X \otimes \underline{\mathbf{G}}_Y \in \mathbb{R}^{R_1Q_1\times\cdots\times R_NQ_N}$ and the factor matrices

$$\mathbf{Z}^{(n)} = \mathbf{X}^{(n)} \boxdot_1 \mathbf{Y}^{(n)} \in \mathbb{R}^{(I_n+J_n-1)\times R_nQ_n}, \tag{20}$$

  where $\mathbf{Z}^{(n)}(:, s_n) = \mathbf{X}^{(n)}(:, r_n) * \mathbf{Y}^{(n)}(:, q_n) \in \mathbb{R}^{(I_n+J_n-1)}$ for $s_n = \overline{r_n q_n} = 1, 2, \ldots, R_nQ_n$.
- **Super Fast discrete Fourier transform** (MATLAB functions fftn($\underline{\mathbf{X}}$) and fft($\mathbf{X}^{(n)}$, [], 1)) of a tensor in the Tucker format

$$\mathcal{F}(\underline{\mathbf{X}}) = \left[\!\left[\underline{\mathbf{G}}_X; \mathcal{F}(\mathbf{X}^{(1)}), \ldots, \mathcal{F}(\mathbf{X}^{(N)})\right]\!\right]. \tag{21}$$

  Note that if the data tensor admits low multilinear rank approximation, then performing the FFT on factor matrices of relatively small size $\mathbf{X}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, instead of a large-scale data tensor, decreases considerably computational complexity. This approach is referred to as the super fast Fourier transform in Tucker format.

  Similar operations can be performed in other TN formats [13].

4 Curse of Dimensionality and Separation of Variables for Multivariate Functions

The term curse of dimensionality was coined by Bellman [55] to indicate that the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with the number of variables, that is, with the dimensionality of the function. In a general context of machine learning and the underlying optimization problems, the "curse of dimensionality" may also refer to an

exponentially increasing number of parameters required to describe the data/system or an extremely large number of degrees of freedom. The term "curse of dimensionality", in the context of tensors, refers to the phenomenon whereby the number of elements, I^N, of an Nth-order tensor of size $(I \times I \times \cdots \times I)$ grows exponentially with the tensor order, N. Tensor volume can therefore easily become prohibitively big for multiway arrays for which the number of dimensions ("ways" or "modes") is very high, thus requiring huge computational and memory resources to process such data. The understanding and handling of the inherent dependencies among the excessive degrees of freedom create both difficult to solve problems and fascinating new opportunities, but comes at a price of reduced accuracy, owing to the necessity to involve various approximations.

The curse of dimensionality can be alleviated or even fully dealt with through tensor network representations; these naturally cater for the excessive volume, veracity and variety of data (see Fig. 1) and are supported by efficient tensor decomposition algorithms which involve relatively simple mathematical operations. Another desirable aspect of tensor networks is their relatively small-scale and low-order core tensors, which act as "building blocks" of tensor networks. These core tensors are relatively easy to handle and visualize, and enable super-compression of the raw, incomplete, and noisy huge-scale data sets. This suggests a solution to a more general quest for new technologies for processing of exceedingly large data sets within affordable computation times [13, 18, 56–58].

To address the curse of dimensionality, this work mostly focuses on approximative low-rank representations of tensors, the so-called low-rank tensor approximations (LRTA) or low-rank tensor network decompositions.

A tensor is said to be in a full or raw format when it is represented as an original (raw) multidimensional array [59], however, distributed storage and processing of high-order tensors in their full format is infeasible due to the curse of dimensionality. The sparse format is a variant of the full tensor format which stores only the nonzero entries of a tensor, and is used extensively in software tools such as the Tensor Toolbox [60] and in the sparse grid approach [61–63].

As already mentioned, the problem of huge dimensionality can be alleviated through various distributed and compressed tensor network formats, achieved by low-rank tensor network approximations.

The underpinning idea is that by employing tensor networks formats, both computational costs and storage requirements may be considerably reduced through distributed storage and computing resources. It is important to note that, except for very special data structures, a tensor cannot be compressed without incurring some compression error, since a low-rank tensor representation is only an approximation of the original tensor.

The concept of compression of multidimensional large-scale data by tensor network decompositions can be intuitively explained as follows [13]. Consider the approximation of an N-variate function $h(\mathbf{x}) = h(x_1, x_2, \ldots, x_N)$ by a finite sum of

products of individual functions, each depending on only one or a very few variables [64–67]. In the simplest scenario, the function $h(\mathbf{x})$ can be (approximately) represented in the following separable form

$$h(x_1, x_2, \ldots, x_N) \cong h^{(1)}(x_1)\, h^{(2)}(x_2) \cdots h^{(N)}(x_N). \tag{22}$$

In practice, when an N-variate function $h(\mathbf{x})$ is discretized into an Nth-order array, or a tensor, the approximation in (22) then corresponds to the representation by rank-1 tensors, also called elementary tensors (see Sect. 2.1). Observe that with I_n, $n = 1, 2, \ldots, N$ denoting the size of each mode and $I = \max_n\{I_n\}$, the memory requirement to store such a full tensor is $\prod_{n=1}^{N} I_n \leq I^N$, which grows exponentially with N. On the other hand, the separable representation in (22) is completely defined by its factors, $h^{(n)}(x_n)$, $(n = 1, 2, \ldots, N)$, and requires only $\sum_{n=1}^{N} I_n \ll I^N$ storage units.

If $x_1, x_2, \ldots, x_N$ are statistically independent random variables, their joint probability density function is equal to the product of marginal probabilities, $p(\mathbf{x}) = p^{(1)}(x_1)p^{(2)}(x_2) \ldots p^{(N)}(x_N)$, in an exact analogy to outer products of elementary tensors. Unfortunately, the form of separability in (22) is rather rare in practice.

It should be noted that a function $h(x_1, x_2)$ is a continuous analogue of a matrix, say $\mathbf{H} \in \mathbb{R}^{I_1 \times I_2}$, while a function $h(x_1, \ldots, x_N)$ in N dimensions is a continuous analogue of an N-order grid tensor $\underline{\mathbf{H}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$. In other words, the discretization of a continuous score function $h(x_1, x_2, \ldots, x_N)$ on a hyper-cube leads to a grid tensor of order N. Specifically, we make use of a grid tensor that approximates and/or interpolates $h(x_1, \ldots, x_N)$ on a grid of points.

The concept of tensor networks rests upon generalized (full or partial) separability of the variables of a high dimensional function. This can be achieved in different tensor formats, including:

1. The Canonical Polyadic (CP) format, where

$$h(x_1, x_2, \ldots, x_N) \cong \sum_{r=1}^{R} h_r^{(1)}(x_1)\, h_r^{(2)}(x_2) \cdots h_r^{(N)}(x_N), \tag{23}$$

in an exact analogy to (22). In a discretized form, the above CP format can be written as an Nth-order tensor

$$\underline{\mathbf{H}} \cong \sum_{r=1}^{R} \mathbf{h}_r^{(1)} \circ \mathbf{h}_r^{(2)} \circ \cdots \circ \mathbf{h}_r^{(N)} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}, \tag{24}$$

where $\mathbf{h}_r^{(n)} \in \mathbb{R}^{I_n}$ denotes a discretized version of the univariate function $h_r^{(n)}(x_n)$, symbol $\circ$ denotes the outer product, and R is the tensor rank.

2. The Tucker format, given by (see Sect. 3.1)

$$h(x_1, \ldots, x_N) \cong \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} g_{r_1,\ldots,r_N}\, h^{(1)}_{r_1}(x_1) \cdots h^{(N)}_{r_N}(x_N), \tag{25}$$

and its distributed tensor network variants,

3. The Tensor Train (TT) format (see Sect. 6.2), in the form

$$\begin{aligned} h(x_1, x_2, \ldots, x_N) \cong \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} h^{(1)}_{r_1}(x_1)\, h^{(2)}_{r_1 r_2}(x_2) \cdots \\ \cdots h^{(N-2)}_{r_{N-2} r_{N-1}}(x_{N-1})\, h^{(N)}_{r_{N-1}}(x_N), \end{aligned} \tag{26}$$

with the equivalent compact matrix representation

$$h(x_1, x_2, \ldots, x_N) \cong \mathbf{H}^{(1)}(x_1)\, \mathbf{H}^{(2)}(x_2) \cdots \mathbf{H}^{(N)}(x_N), \tag{27}$$

where $\mathbf{H}^{(n)}(x_n) \in \mathbb{R}^{R_{n-1} \times R_n}$, with $R_0 = R_N = 1$.

All the above approximations adopt the form of "sum-of-products" of single-dimensional functions, a procedure which plays a key role in all tensor factorizations and decompositions.

Indeed, in many applications based on multivariate functions, a relatively good approximations are obtained with a surprisingly small number of factors; this number corresponds to the tensor rank, R, or tensor network ranks, $\{R_1, R_2, \ldots, R_N\}$ (if the representations are exact and minimal). However, for some specific cases this approach may fail to obtain sufficiently good low-rank TN approximations [67]. The concept of generalized separability has already been explored in numerical methods for high-dimensional density function equations [22, 66, 67] and within a variety of huge-scale optimization problems [13, 14].

To illustrate how tensor decompositions address excessive volumes of data, if all computations are performed on a CP tensor format in (24) and not on the raw Nth-order data tensor itself, then instead of the original, exponentially growing, data dimensionality of I^N, the number of parameters in a CP representation reduces to NIR, which scales linearly in the tensor order N and size I. For example, the discretization of a 5-variate function over 100 sample points on each axis would yield the difficulty to manage $100^5 = 10{,}000{,}000{,}000$ sample points, while a rank-2 CP representation would require only $5 \times 2 \times 100 = 1000$ sample points.

In contrast to CP decomposition algorithms, TT tensor network formats in (26) exhibit both very good numerical properties and the ability to control the error of approximation, so that a desired accuracy of approximation is obtained relatively easily [13, 68–70]. The main advantage of the TT format over the CP decomposition is the ability to provide stable quasi-optimal rank reduction, achieved through, for example, truncated singular value decompositions (tSVD) or adaptive

cross-approximation [64, 71, 72]. This makes the TT format one of the most stable and simple approaches to separate latent variables in a sophisticated way, while the associated TT decomposition algorithms provide full control over low-rank TN approximations.[4] We therefore, make extensive use of the TT format for low-rank TN approximations and employ the TT toolbox software for efficient implementations [68]. The TT format will also serve as a basic prototype for high-order tensor representations, while we also consider the Hierarchical Tucker (HT) and the Tree Tensor Network States (TTNS) formats (having more general tree-like structures) whenever advantageous in applications [13].

Furthermore, the concept of generalized separability of variables and the tensorization of structured vectors and matrices allows us to to convert a wide class of huge-scale optimization problems into much smaller-scale interconnected optimization sub-problems which can be solved by existing optimization methods [11, 14].

The tensor network optimization framework is therefore performed through the two main steps:

- Tensorization of data vectors and matrices into a high-order tensor, followed by a distributed approximate representation of a cost function in a specific low-rank tensor network format.
- Execution of all computations and analysis in tensor network formats (i.e., using only core tensors) that scale linearly, or even sub-linearly (quantized tensor networks), in the tensor order N. This yields both the reduced computational complexity and distributed memory requirements.

The challenge is to extend beyond the standard Tucker and CP tensor decompositions, and to demonstrate the perspective of TNs in extremely large-scale data analytic, together with their role as a mathematical backbone in the discovery of hidden structures in prohibitively large-scale data. Indeed, TN models provide a framework for the analysis of linked (coupled) blocks of tensors with millions and even billions of non-zero entries [13, 14].

5 Tensor Networks Approaches for Deep Learning

Revolution (breakthroughs) in the fields of Artificial Intelligence (AI) and Machine Learning triggered by class of deep convolutional neural networks (DCNNs), often simply called CNNs, has been a vehicle for a large number of practical applications and commercial ventures in computer vision, speech recognition, language processing, drug discovery, biomedical informatics, recommender systems, robotics, games, and artificial creativity, to mention just a few.

[4] Although similar approaches have been known in quantum physics for a long time, their rigorous mathematical analysis is still a work in progress (see [27, 69] and references therein).

The renaissance of deep learning neural networks [5, 6, 73, 74] has both created an active frontier of machine learning and has provided many advantages in applications, to the extent that the performance of DNNs in multi-class classification problems can be similar or even better than what is achievable by humans.

Deep learning is highly interesting in very large-scale data analysis for many reasons, including the following [14]:

1. High-level representations learnt by deep NN models, that are easily interpretable by humans, can also help us to understand the complex information processing mechanisms and multi-dimensional interaction of neuronal populations in the human brain;
2. Regarding the degree of nonlinearity and multi-level representation of features, deep neural networks often significantly outperform their shallow counterparts;
3. In big data analytic, deep learning is very promising for mining structured data, e.g., for hierarchical multi-class classification of a huge number of images.

It is well known that both shallow and deep NNs are universal function approximators in the sense that they are able to approximate arbitrarily well any continuous function of N variables on a compact domain, under the condition that a shallow network has an unbounded width (i.e., the size of a hidden layer), that is, an unlimited number of parameters. In other words, a shallow NN may require a huge (intractable) number of parameters (curse of dimensionality), while DNNs can perform such approximations using a much smaller number of parameters.

Universality refers to the ability of a deep learning network to approximate any function when no restrictions are imposed on its size. On the other hand, depth efficiency refers to the case when a function realized by polynomially-sized deep neural network requires shallow networks to have super-polynomial (exponential) size for the same accuracy of approximation (course of dimensionality). This is often referred to as the expressive power of depth.

Despite recent advances in the theory of DNNs, there are several open fundamental challenges (or open problems) related to understanding high performance DNNs, especially the most successful and perspective DCNNs [13, 14]:

- Theoretical and practical bounds on the expressive power of a specific architecture, i.e., quantification of the ability to approximate or learn wide classes of unknown nonlinear functions;
- Ways to reduce the number of parameters without a dramatic reduction in performance;
- Ability to generalize while avoiding overfitting in the learning process;
- Fast learning and the avoidance "bad" local and spurious minima, especially for highly nonlinear score (objective) functions;
- Rigorous investigation of the conditions under which deep neural networks are "much better" the shallow networks (i.e., NNs with one hidden layer).

The aim of this section is to discuss the many advantages of tensor networks in addressing the first two of the above challenges and to build up both intuitive

and mathematical links between DNNs and TNs. Revealing such links and inherent connections will both cross-fertilize deep learning and tensor networks and provide new insights.

In addition to establishing the existing and developing new links, this will also help to optimize existing DNNs and/or generate new architectures with improved performances.

We shall first present an intuitive approach using a simplified hierarchical Tucker (HT) model, followed by alternative simple but efficient, tensor train/tensor chain (TT/TC) architectures. We also propose to use more sophisticated TNs, such as MERA tensor network models in order to enable more flexibility, improved performance, and/or higher expressive power of the next generation of DCCNs.

5.1 Why Tensor Networks Are Important in Deep Learning?

Several research groups have recently investigated the application of tensor decompositions to simplify DNNs and to establish links between the deep learning and low-rank tensor networks [14, 75–80]. For example, [80] presented a general and constructive connection between Restricted Boltzmann Machines (RBM), which is a fundamental basic building block in class of Deep Boltzmann Machines, and (TNs) together with the correspondence between general Boltzmann machines and TT/MPS. In a series of research papers [30, 79, 81, 82] the authors analyze the expressive power of a class of DCNNs using simplified Hierarchal Tucker (HT) models (see the next sections). Particularly, Convolutional Arithmetic Circuits (ConvAC), also known as Sum-Product Networks, and Convolutional Rectifier Networks (CRN) have been considered as HT model. They claim that a shallow (single hidden layer) network realizes the classic CP decomposition, whereas a deep network with $\log_2 N$ hidden layers realizes Hierarchical Tucker (HT) decomposition (see the next section). Some researchers also argued that the "unreasonable success" of deep learning can be explained by inherent law of physics, such as the theory of TNs that often employ physical constraints locality, symmetry, compositional hierarchical functions, entropy, and polynomial log-probability, imposed on measurements or input training data [77, 80, 83]. In fact, a very wide spectrum of tensor networks can be potentially used to model and analyze some specific classes of DNNs, in order to obtain simpler and/or more efficient neural networks in the sense that they could provide more expressive power or reduced complexity. Such an approach not only promises to open the door to various mathematical and algorithmic tools for enhanced analysis of DNNs, but also allows us to design novel multi-layer architectures and practical implementations of various deep learning systems. In other words, the consideration of tensor networks in this context may give rise to new NN architectures which could be even potentially superior to the existing ones, but have so far been overlooked by practitioners. Furthermore, methods used for reducing or approximating TNs could be a vehicle to achieve more efficient DNNs, with a reduced number of parameters. This follows from the facts that redundancy

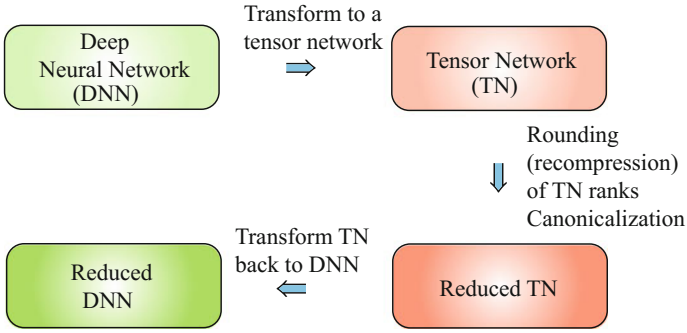

Fig. 9 Optimization of Deep Neural Networks (DNNs) using The Tensor Networks (TNs) approach. In order to simplify a specific DNN and reduce the number of its parameters, we first transform the DNN into a specific TN, e.g., TT/MPS, then transform the approximated (with reduced rank) TN back to a new optimized DNN. Such transformation can performed in a layer by layer fashion, or globally for the whole DNN. For detailed discussions of such mappings for the Restricted Boltzmann Machine (RBM) see [80]. Optionally, we may choose to first construct and learn (e.g., via tensor completion) a tensor network and then transform it to an equivalent DNN

is inherent both in TNs and DNNs. Moreover, both TNs and DNNs are usually not unique. For example, two NNs with different connection weights and biases may result into the modeling the same nonlinear function. Therefore, the knowledge about redundancy in TNs can help simplify DNNs [14].

The general concept of optimization of DNNs via TNs is illustrated in Fig. 9. Given a specific DNN, we first construct an equivalent TN representation of the given DNN, then the TN is transformed into its reduced or canonical form by performing, e.g., the truncated SVD at each rank (bond). This will reduce the rank dimensions to the minimal requirement determined by a desired accuracy of approximation. Finally, we map back the reduced and optimized TN to another DNN. Since a rounded (approximated) TN has smaller ranks dimensions, a final DNN can be simpler than the original one,and with the same or slightly reduced performance.

It should be noted that, in practice, low-rank TN approximations have many potential advantages over a direct reduction of redundant DNNs, due to availability of many efficient optimization methods to reduce the number of parameters and achieve a pre-specified approximation error. Moreover, low-rank tensor networks are capable of avoiding the curse of dimensionality through low-order sparsely interconnected core tensors.

In the past two decades, quantum physicists and computer scientists have developed solid theoretical understanding and efficient numerical techniques for low-rank TN decompositions.

The entanglement entropy, Renyi's entropy, entanglement spectrum and long range correlations are four of the most widely used quantities (calculated from a spatial reduced density matrix) investigated in the theory of tensor networks. The spatial reduced density matrix is determined by splitting a TN into two parts, say, regions A and B, where a density matrix in region A is given by integrating out all

the degrees of freedom in region B. The entanglement spectra are determined by the eigenvalues of the reduced density matrix [84, 85].

Entanglement is a physical phenomenon that occurs when pairs or groups of particles, such as photons, electrons, or qubits, are generated or interact in such way that the quantum state of each particle cannot be described independently of the others, so that a quantum state must be described for the system as a whole. Entanglement entropy is therefore a measure for the amount of entanglement. Strictly speaking, entanglement entropy is a measure of how quantum information is stored in a quantum state and it is mathematically expressed as the von Neumann entropy of the reduced density matrix. Entanglement entropy characterizes the information content of a bipartition of a specific TN. Furthermore, the entanglement area law explains that the entanglement entropy increases only proportionally to the boundary between the two tensor sub-networks. Also entanglement entropy characterizes the information content of the distribution of singular values of a matricized tensor, and can be viewed as a proxy for the correlations between the two partitions; uncorrelated data has zero entanglement entropy at any bipartition.

Note that TNs are usually designed to efficiently represent large systems which exhibit a relatively low entanglement entropy. In practice, we often need to only care about a small fraction of the input training data among a huge number of possible inputs similar to deep neural networks. This all suggest that certain guiding principles in DNNs correspond to the entanglement area law used in the theory of tensor networks. These may then used to quantify the expressive power of a wide class of DCNNs. Note that long range correlations also typically increase with the entanglement. We therefore conjecture that realistic data sets in most successful machine learning applications have relatively low entanglement entropies [86]. On the other hand, by exploiting the entanglement entropy bound of TNs, we can rigorously quantify the expressive power of a wide class of DNNs applied to complex and highly correlated data sets.

5.2 Basic Features of Deep Convolutional Neural Networks

Basic DCNNs are usually characterized by at least three features: locality, weight sharing (optional) and pooling explained below [14, 79]

- Locality refers to the connection of a (artificial) neuron only to neighboring neurons in the preceding layer, as opposed to being fed by the entire layer (this is consistent with biological NNs).
- Weight sharing reflects the property that different neurons in the same layer, connected to different neighborhoods in the preceding layer, often share the same weights. Note that weight sharing, when combined with locality, gives rise to standard convolution. However, it should noted that although weight sharing may reduce the complexity of a deep neural network, it is optional. However, the locality

at each layer is a key factor which gives DCNNs an exponential advantage over shallow NNs [77, 87, 88].

- Pooling, is essentially an operator that gradually decimates (reduces) layer sizes by replacing the local population of neural activations in a spatial window by a single value (e.g., by taking their maxima, average values or their scaled products). In the context of images, pooling induces invariance to translation, which often does not affect semantic content, and is interpreted as a way to create a hierarchy of abstractions in the patterns that neurons respond to [14, 79, 87].

Usually, DCNNs perform much better when dealing with compositional function approximations[5] and multi-class classification problems than shallow network architectures with one hidden layer. In fact, DCNNs can efficiently and conveniently select a subset of features for multiple classes, while for efficient learning a DCNN model can be pre-trained by first learning each DCNN layer, followed by fine tuning of the parameter of the entire model e.g., stochastic gradient descent. To summarize, the deep learning neural networks have the ability to exploit and approximate the complexity of compositional hierarchical functions arbitrarily well, whereas shallow networks are blind to them.

5.3 *Score Functions for Deep Convolutional Neural Networks*

Consider a multi-class classification task where the input training data, also called local structures or instances (e.g., input patches in images), are denoted by $X = (\mathbf{x}_1, \dots, \mathbf{x}_N)$, where $\mathbf{x}_n \in \mathbb{R}^S \;\; n = 1, \dots, N)$ belonging to one of C categories (classes) denoted by $y_c \in \{y_1, y_2, \dots, y_C\}$. Such a representation is quite natural for many high-dimensional data—in images, the local structures represent patches consisting of S pixels, while in audio data voice can be represented through spectrograms.

For this kind of problems, DCNNs aim is to model the following set of multivariate score functions:

$$
\begin{aligned}
h_{y_c}(\mathbf{x}_1, \dots, \mathbf{x}_N) &= \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \underline{\mathbf{W}}_{y_c}(i_1, \dots, i_N)\; \boldsymbol{\Phi}_{i_1,\dots,i_N}(\mathbf{x}_1, \dots, \mathbf{x}_N), \\
\boldsymbol{\Phi}_{i_1,\dots,i_N}(\mathbf{x}_1, \dots, \mathbf{x}_N) &= \prod_{n=1}^{N} f_{\theta_{i_n}}(\mathbf{x}_n), \qquad (28) \\
\text{for} \quad y_c &= y_1, y_2, \dots, y_C,
\end{aligned}
$$

where $\underline{\mathbf{W}}_{y_c} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is an Nth-order coefficient tensor (typically, with all dimensions $I_n = I, \;\forall n$), N is the number of (typically overlapped) input patches $\mathbf{x}_n$, I_n is

[5] A compositional function can take, for example, the following form $h_1(\dots h_3(h_{21}(h_{11}(x_1, x_2)h_{12}(x_3, x_4)), h_{22}(h_{13}(x_5, x_6)h_{14}(x_7, x_8)) \dots)))$.

the size (dimension) of each mode $\underline{\mathbf{W}}_{y_c}$, and $f_{\theta_1}, \ldots, f_{\theta_{I_n}}$ are referred to as the representation functions (in the representation layer) selected from a parametric family of nonlinear functions.[6]

In general, the one-dimensional basis functions could be polynomials, splines or other sets of basis functions. Natural choices for this family of nonlinear functions are also radial basis functions (Gaussian RBFs), wavelets, and affine functions followed by point-wise activations. Particularly interesting are Gabor wavelets, owing to their ability to induce features that resemble representations in the visual cortex of human brain.

Note that the representation functions in standard (artificial) neurons have the form

$$f_{\theta_i}(\mathbf{x}) = \sigma(\tilde{\mathbf{w}}_i^{\mathrm{T}}\mathbf{x} + b_i), \tag{29}$$

for the set of parameters $\theta_i = \{\tilde{\mathbf{w}}_i, b_i\}$, where $\sigma(\cdot)$ is a suitably chosen activation function.

The representation layer play a key role to transform the inputs, by means of I nonlinear functions, $f_{\theta_i}(\mathbf{x}_n)$ $(i = 1, 2, \ldots, I)$, to template input patches, thereby creating I feature maps [81]. Note that the representation layer can be expressed by a feature vector defined as

$$\mathbf{f} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_1) \otimes \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_2) \otimes \cdots \otimes \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_N) \in \mathbb{R}^{I_1 I_2 \ldots I_N}, \tag{30}$$

where $\mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_n) = [f_{\theta_1}(\mathbf{x}_n), f_{\theta_2}(\mathbf{x}_n), \ldots, f_{\theta_{I_n}}(\mathbf{x}_n)]^T \in \mathbb{R}^{I_n}$ for $n = 1, 2, \ldots, N$ and $i_n = 1, 2, \ldots, I_n$.

Alternatively, the representation layer can be expressed as rank one tensor (see Fig. 10a)

$$\underline{\mathbf{F}} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_1) \circ \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_2) \circ \cdots \circ \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_N) \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}. \tag{31}$$

This allows us to represent the score function as an inner product of two tensors, as illustrated in Fig. 10a

$$h_{y_c}(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \langle \underline{\mathbf{W}}_{y_c}, \underline{\mathbf{F}} \rangle = \underline{\mathbf{W}}_{y_c} \bar{\times}_1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_1) \bar{\times}_2 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_2) \ldots \bar{\times}_N \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_N). \tag{32}$$

To simplify the notations of a grid tensor, we can construct square matrices $\mathbf{F}_n$ $(n = 1, 2, \ldots, N)$, as follows

$$\mathbf{F}_n = \begin{bmatrix} f_{\theta_1}(\mathbf{x}_n^{(1)}) & f_{\theta_2}(\mathbf{x}_n^{(1)}) & \ldots & f_{\theta_{I_n}}(\mathbf{x}_n^{(1)}) \\ f_{\theta_1}(\mathbf{x}_n^{(2)}) & f_{\theta_2}(\mathbf{x}_n^{(2)}) & \ldots & f_{\theta_{I_n}}(\mathbf{x}_n^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ f_{\theta_1}(\mathbf{x}_n^{(I_n)}) & f_{\theta_2}(\mathbf{x}_n^{(I_n)}) & \ldots & f_{\theta_{I_n}}(\mathbf{x}_n^{(I_n)}) \end{bmatrix} \in \mathbb{R}^{I_n \times I_n}, \tag{33}$$

[6] Note that the representation layer can be considered as a tensorization of input patches $\mathbf{x}_n$.

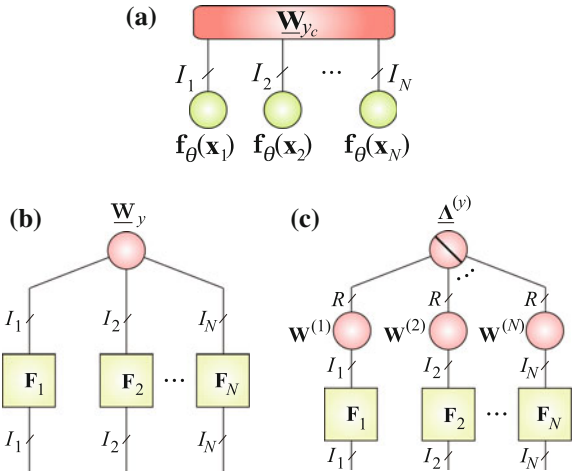

Fig. 10 Various representations of the score function of a DCNN. **a** Direct Representation of the score function $h_{y_c}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = \underline{\mathbf{W}}_{y_c} \bar{\times}_1 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_1) \bar{\times}_2 \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_2) \cdots \bar{\times}_N \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}_N)$. Note that the coefficient tensor $\underline{\mathbf{W}}_c$ can be represented in a distributed form by any suitable tensor network. **b** Graphical illustration of the Nth-order grid tensor of the score function h_c. This model can be considered as a special case of Tucker-N model where the representation matrix $\mathbf{F}_n \in \mathbb{R}^{I_n \times I_n}$ built up factor matrices; note that typically all the factor matrices are the same and $I_n = I,\ \forall n$. **c** CP decomposition of the coefficient tensor $\underline{\mathbf{W}}_{y_c} = \underline{\mathbf{\Lambda}}^{(y_c)} \times_1 \mathbf{W}^{(1)} \times_2 \mathbf{W}^{(2)} \ldots \times_N \mathbf{W}^{(N)} = \sum_{r=1}^{R} \lambda_r^{(y_c)} (\mathbf{w}_r^{(1)} \circ \mathbf{w}_r^{(2)} \circ \cdots \circ \mathbf{w}_r^{(N)})$, where $\mathbf{W}^{(n)} = [\mathbf{w}_1^{(n)}, \ldots, \mathbf{w}_R^{(n)}] \in \mathbb{R}^{I_n \times R}$. This CP model corresponds to a simple shallow neural network with one hidden layer, comprising weights $w_{ir}^{(n)}$, and the output layer comprising weights $\lambda_r^{(y_c)}$, $r = 1, \ldots, R$

which holding the values of taken by the nonlinear basis functions $\{f_{\theta_1}, \ldots, f_{\theta_{I_n}}\}$ on the selected fixed vectors, referred to as templates, $\{\mathbf{x}_n^{(1)}, \mathbf{x}_n^{(2)}, \ldots, \mathbf{x}_n^{(I_n)}\}$. Usually, we can assume that $I_n = I,\ \forall n$, and $\mathbf{x}_n^{(i_n)} = \mathbf{x}^{(i)}$ [30].

For discrete data values, the score function can be represented by a grid tensor, as graphically illustrated in Fig. 10b. The grid tensor of the nonlinear score function $h_{y_c}(\mathbf{x}_1, , \ldots, \mathbf{x}_N)$ determined over all the templates $\mathbf{x}_n^{(1)}, \mathbf{x}_n^{(2)}, \ldots, \mathbf{x}_n^{(I)}$ can be expressed as a grid tensor

$$\underline{\mathbf{W}}(h_{y_c}) = \underline{\mathbf{W}}_{y_c} \times_1 \mathbf{F}_1 \times_2 \mathbf{F}_2 \cdots \times_N \mathbf{F}_N. \tag{34}$$

Of course, since the order N of the coefficient (core) tensor is large, it cannot be implemented, or even saved on computer due to the curse of dimensionality.

The simplest model to represent coefficient tensors would be to apply the CP decomposition to reduce the number of parameters, as illustrated in Fig. 10c. This leads to a simple shallow network, however, this approach is associated with two problems: (i) the rank R of the coefficient tensor $\underline{\mathbf{W}}_{y_c}$ can be very large (so compression

of parameters cannot be very high), (ii) the existing CP decomposition algorithms are not very stable for very high-order tensors, and so an alternative promising approach would be to apply tensor networks such as HT that enable us to avoid the curse of dimensionality.

Following the representation layer, a DCNN may consists of a cascade of L convolutional hidden layers with pooling in-between, where the number of layers L should be at least two. In other words, each hidden layer performs 3D or 4D convolution followed by spatial window pooling, in order to reduce (decimate) feature maps by e.g., taking a product of the entries in sub-windows. The output layer is a linear dense layer.

Classification can then be carried out in a standard way, through the maximization of a set of labeled score functions, h_{y_c} for C classes, that is, the predicted label for the input instants $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ will be the index $\hat{y}_c$ for which the score value attains a maximum, that is

$$\hat{y}_c = \underset{y_c \in \{y_1, \ldots, y_C\}}{\arg\max} \; h_{y_c}(\mathbf{x}_1, \ldots, \mathbf{x}_N). \tag{35}$$

Such score functions can be represented through their coefficient tensors which, in turn, can be approximated by low-rank tensor network decompositions [13].

The one restriction of the so formulated score functions (29) is that they allow for straightforward implementation of only a particular class of DCNNs, called convolutional Arithmetic Circuit (ConvAC). However, the score functions can be approximated indirectly and almost equivalently using more popular CNNs (see the next section). For example, it was shown recently how NNs with a univariate ReLU nonlinearity may perform multivariate function approximation [77].

The main idea is to employ a low-rank tensor network representation to approximate and interpolate a multivariate function $h_{y_c}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ of N variables by a finite sum of separated products of simpler functions (i.e., via sparsely interconnected core tensors) [13, 14].

6 Convolutional Arithmetic Circuits (ConvAC) Using Tensor Networks

Once the set of score functions has been formulated (29), we need to construct (design) a suitable multilayered or distributed representation for DCNN implementation. The objective is to estimate the parameters $\theta_1, \ldots, \theta_I$ and coefficient tensors[7] $\underline{\mathbf{W}}_{y_1}, \ldots, \underline{\mathbf{W}}_{y_C}$. Since the tensors are of Nth-order and each with I^N entries, in order to avoid the curse of dimensionality, we need to perform dimensionality reduction

[7] It should be noted that these tensors share the same entries, except for the parameters in the output layer.

through low-rank tensor network decompositions. Note that a direct implementation of (29) is intractable due to a huge number of parameters.

Conceptually, the ConvAC can be divided into three parts: (i) the first (input) layer is the representation layer which transforms input vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ into $N \cdot I$ real valued scalars $\{f_{\theta_i}(\mathbf{x}_n)\}$ for $n = 1, \ldots, N$ and $i = 1, \ldots, I$. In other words, the representation functions, $f_{\theta_i} : \mathbb{R}^S \to \mathbb{R}$, $i = 1, \ldots, I$, map each local patch $\mathbf{x}_n$ into a feature space of dimension I. We can denote the feature vector by $\mathbf{f}_n = [f_{\theta_1}(\mathbf{x}_n), \ldots, f_{\theta_I}(\mathbf{x}_n)]^{\mathrm{T}} \in \mathbb{R}^I$, $n = 1, \ldots, N$; (ii) the second, a key or kernel part, is a convolutional arithmetic circuits with many hidden layers that takes the $N \cdot I$ measurements (training samples) generated by the representation layer; (iii) the output layer represented by a full matrix $\mathbf{W}^{(L)}$, which computes C different score functions h_{y_c} [79].

6.1 Hierarchical Tucker (HT) and Tree Tensor Network State (TTNS) Models

The simplified HT tensor network [79] shown in Fig. 11 contains sparse 3rd-order core tensors $\underline{\mathbf{W}}^{(l,j)} \in \mathbb{R}^{R^{(l-1,2j-1)} \times R^{(l-1,2j-1)} \times R^{(l,j)}}$ for $l = 1, \ldots, L-1$ and matrices $\mathbf{W}^{(0,j)} = [\mathbf{w}_1^{(0,j)}, \ldots, \mathbf{w}_{R^{(0,j)}}^{(0,j)}] \in \mathbb{R}^{I_j \times R^{(0,j)}}$ for $l = 0$ and $j = 1, \ldots, N/2^l$, and a full matrix $\mathbf{W}^{(L)} = [\mathbf{w}_1^{(L)}, \ldots, \mathbf{w}_{y_C}^{(L)}] \in \mathbb{R}^{R^{(L-1)} \times y_C}$ with column vectors $\mathbf{w}_{y_c}^{(L)} = \mathrm{diag}(\mathbf{W}_{y_c}^{(L)}) = [\lambda_1^{(y_c)}, \ldots, \lambda_{R^{(L-1)}}^{(y_c)}]^T$ in the output L-layer (or equivalently the output sparse tensor $\underline{\mathbf{W}}^{(L)} \in \mathbb{R}^{R^{(L-1)} \times R^{(L-1)} \times y_C}$. The number of channels in the input layer is denoted by I, while for the jth node in the lth layer (for $l = 0, 1, \ldots, L-1$) is denoted by $R^{(l,j)}$. The y_C different values of score functions are calculated in the output layer.

For simplicity, and in order to mimic basic features of the standard ConvAC, we assume that $R^{(l,j)} = R^{(l)}$ for all j, and that frontal slices of the core tensors $\underline{\mathbf{W}}^{(l,j)}$ are diagonal matrices with entries $w_{r^{(l-1)},r^{(l)}}^{l,j}$. Note that such sparse core tensors can be represented by non-zero matrices defined as $\mathbf{W}^{(l,j)} \in \mathbb{R}^{R^{(l-1)} \times R^{(l)}}$.

The simplified HT tensor network can be mathematically described in the following recursive form

$$
\begin{aligned}
\mathbf{W}^{(0,j)} &= [\mathbf{w}_1^{(0,j)}, \ldots, \mathbf{w}_{R^{(0)}}^{(0,j)}] \in \mathbb{R}^{I_j \times R^{(0)}} \\
\mathbf{W}_{r^{(1)}}^{(\le 1,j)} &= \sum_{r^{(0)}=1}^{R^{(0)}} w_{r^{(0)},r^{(1)}}^{(1,j)} \cdot \left(\mathbf{w}_{r^{(0)}}^{(0,2j-1)} \circ \mathbf{w}_{r^{(0)}}^{(0,2j)} \right) \in \mathbb{R}^{I_{2j-1} \times I_{2j}} \\
&\ldots \\
\underline{\mathbf{W}}_{r^{(l)}}^{(\le l,j)} &= \sum_{r^{(l-1)}=1}^{R^{(l-1)}} w_{r^{(l-1)},r^{(l)}}^{(l,j)} \cdot \left(\underline{\mathbf{W}}_{r^{(l-1)}}^{(\le l-1,2j-1)} \circ \underline{\mathbf{W}}_{r^{(l-1)}}^{(\le l-1,2j)} \right)
\end{aligned}
\tag{36}
$$

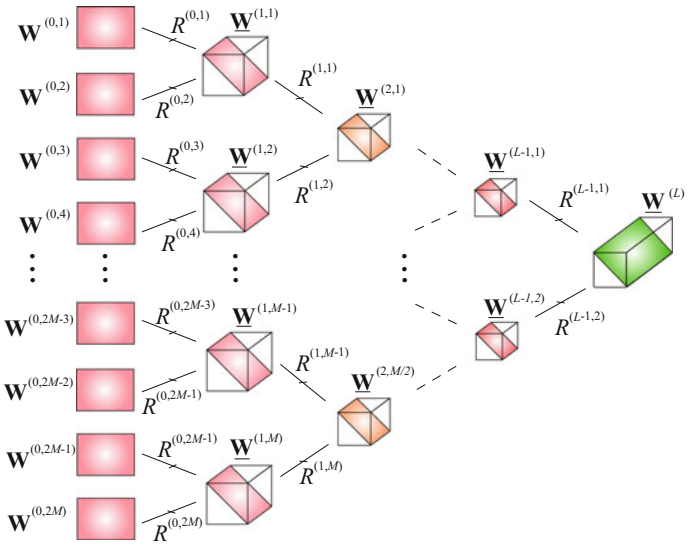

Fig. 11 Architecture of a simplified Hierarchical Tucker (HT) network with sparse core tensors, which simulates the coefficient tensor for a ConvAC deep learning network with a pooling-2 window [79]. The HT tensor network consists of $L = \log_2(N)$ hidden layers and pooling-2 window. For simplicity, we assumed that we, $N = 2M = 2^L$ input patches, $R^{(l,j)} = R^{(l)}$), for $l = 0, 1, \ldots, L-1$. The representation layer is not shown explicitly in this figure. Note that since all core tensors can be represented by matrices, we do not need to use tensors notation in this case

$$\underline{\mathbf{W}}^{(\le L-1,j)}_{r^{(L-1)}} = \sum_{r^{(L-2)}=1}^{R^{(L-2)}} w^{(L-1,j)}_{r^{(L-2)},r^{(L-1)}} \cdot \left(\underline{\mathbf{W}}^{(\le L-2,2j-1)}_{r^{(L-2)}} \circ \underline{\mathbf{W}}^{(\le L-2,2j)}_{r^{(L-2)}} \right)$$

$$\underline{\mathbf{W}}_{y_c} \cong \sum_{r^{(L-1)}=1}^{R^{(L-1)}} \lambda^{(y_c)}_{r^{(L-1)}} \cdot \left(\underline{\mathbf{W}}^{(\le L-1,1)}_{r^{(L-1)}} \circ \underline{\mathbf{W}}^{(\le L-1,2)}_{r^{(l-1)}} \right) \in \mathbb{R}^{I_1 \times \cdots \times I_N},$$

where $\boldsymbol{\lambda}^{(y_c)} = \text{diag}(\lambda^{(y_c)}_1, \ldots, \lambda^{(y_c)}_{R^{(L-1)}}) = \underline{\mathbf{W}}^{(L)}(:,:,y_c)$.

In a special case when the weights in each layer are shared, i.e., $\mathbf{W}^{(l,1)} = \mathbf{W}^{(l,2)} = \cdots = \mathbf{W}^{(l)}$, the above equation can be considerably simplified to

$$\underline{\mathbf{W}}^{\le l}_{r^{(l)}} = \sum_{r^{(l-1)}=1}^{R^{(l-1)}} w^{(l)}_{r^{(l-1)},r^{(l)}} \, (\underline{\mathbf{W}}^{\le l-1}_{r^{(l-1)}} \circ \underline{\mathbf{W}}^{\le l-1}_{r^{(l-1)}}) \tag{37}$$

for the layers $l = 1, \ldots, L-1$, while for the output layer

$$\underline{\mathbf{W}}_{y_c} \cong \sum_{r^{(L-1)}=1}^{R^{(L-1)}} \lambda^{(y_c)}_{r^{(L-1)}} (\underline{\mathbf{W}}^{\le L-1}_{r^{(L-1)}} \circ \underline{\mathbf{W}}^{\le L-1}_{r^{(L-1)}}), \tag{38}$$

where $\underset{r^{(l)}}{\mathbf{W}}^{\leq l} = \underline{\mathbf{W}}^{\leq l}(:, \ldots, :, r^{(l)}) \in \mathbb{R}^{I \times \cdots \times I}$ are sub-tensors of $\underline{\mathbf{W}}^{\leq l}$, for each $r^{(l)} = 1, \ldots, R^{(l)}$, and $w^{(l)}_{r^{(l-1)}, r^{(l)}}$ is the $(r^{(l-1)}, r^{(l)})$th entry of the weight matrix $\mathbf{W}^{(l)} \in \mathbb{R}^{R^{(l-1)} \times R^{(l)}}$.

However, it should be noted that the simplified HT model shown in Fig. 11 has a limited ability to approximate an arbitrary coefficient tensor, $\underline{\mathbf{W}}_{y_c}$, due to strong constraints imposed of core tensors. A more flexible and powerful model is shown in Fig. 12, in which constraints imposed on 3rd-order cores have been completely removed. Such a HT tensor network (with a slight abuse of notation) can be mathematically expressed as

$$\begin{aligned} \mathbf{W}^{(\leq 1,j)}_{r^{(1)}} &= \sum_{r_1=1}^{R^{(0,2j-1)}} \sum_{r_2=1}^{R^{(0,2j)}} w^{(1,j)}_{r_1,r_2,r^{(1)}} \cdot \left(\mathbf{w}^{(0,2j-1)}_{r_1} \circ \mathbf{w}^{(0,2j)}_{r_2} \right) \\ &\cdots \end{aligned} \tag{39}$$

$$\begin{aligned} \underset{r^{(l)}}{\mathbf{W}}^{(\leq l,j)} &= \sum_{r_1=1}^{R^{(l-1,2j-1)}} \sum_{r_2=1}^{R^{(l-1,2j)}} w^{(l,j)}_{r_1,r_2,r^{(l)}} \cdot \left(\underset{r^{(l-1)}}{\mathbf{W}}^{(\leq l-1,2j-1)} \circ \underset{r^{(l-1)}}{\mathbf{W}}^{(\leq l-1,2j)} \right) \\ &\cdots \\ \underset{r^{(L-1)}}{\mathbf{W}}^{(\leq L-1,j)} &= \sum_{r_1=1}^{R^{(L-2,2j-1)}} \sum_{r_2=1}^{R^{(L-2,2j)}} w^{(L-1,j)}_{r_1,r_2,r^{(L-1)}} \cdot \left(\underset{r^{(L-2)}}{\mathbf{W}}^{(\leq L-2,2j-1)} \circ \underset{r^{(L-2)}}{\mathbf{W}}^{(\leq L-2,2j)} \right) \end{aligned}$$

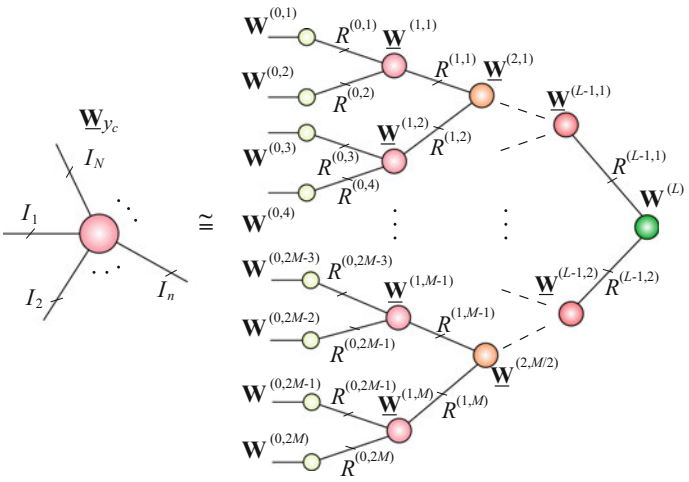

Fig. 12 Hierarchical Tucker (HT) tensor network for the approximation of coefficient tensors, $\underline{\mathbf{W}}_{y_c}$, of the score functions $h_{y_c}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$

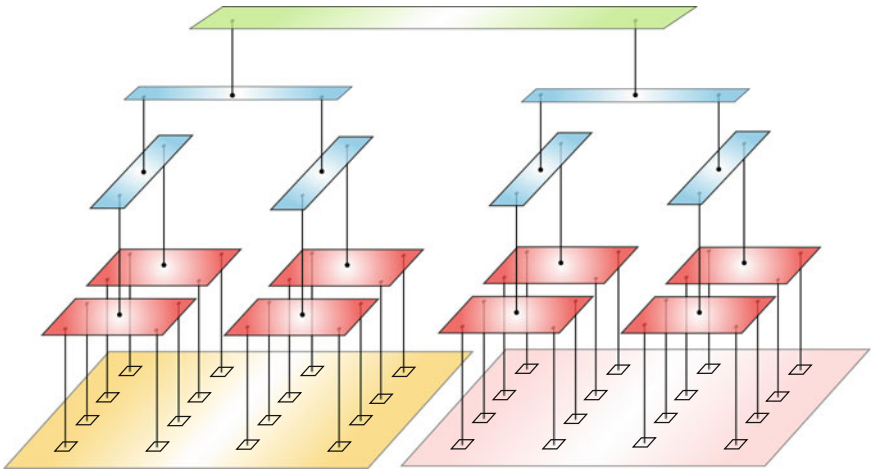

Fig. 13 Tree Tensor Networks States (TTNS) with variable order of core tensors. The rectangles represent core tensors of orders 5 and 3 that allows pooling of window size 4 and 2, respectively

$$\underline{\mathbf{W}}^{(y_c)} \cong \sum_{r_1=1}^{R^{(L-1,2j-1)}} \sum_{r_2=1}^{R^{(L-1,2j)}} w^{(L)}_{r_1,r_2,y_c} \cdot \left(\underline{\mathbf{W}}^{(\leq L-1,1)}_{r^{(L-1)}} \circ \underline{\mathbf{W}}^{(\leq L-1,2)}_{r^{(l-1)}} \right).$$

The HT network can be further extended to the Tree Tensor Networks States (TTNS), as illustrated in Fig. 13. The use of TTNS, instead HT tensor networks, allows for more flexibility in the choice of size of pooling-window, the pooling size in each hidden layer cab be adjusted by applying core tensors with a suitable variable order in each layer. For example, if we use 5th-order (4rd-order) core tensors instead 3rd-order cores, then the pooling will employ a size-4 pooling window (size-3 pooling) instead of only size-2 pooling window when using 3rd-order core tensors in HT tensor networks. For more detail regrading HT networks and their generalizations to TTNS [13].

6.2 *Alternative Tensor Network Model: Tensor Train (TT) Networks*

We should emphasize that the HT/TTNS architectures are not the only one suitable TN decompositions which can be used to model DCNNs, and the whole family of powerful tensor networks can be employed to model individual hidden layers. In this section we discuss modified TT/MPS and TC models for this purpose for which efficient learning algorithms exist.

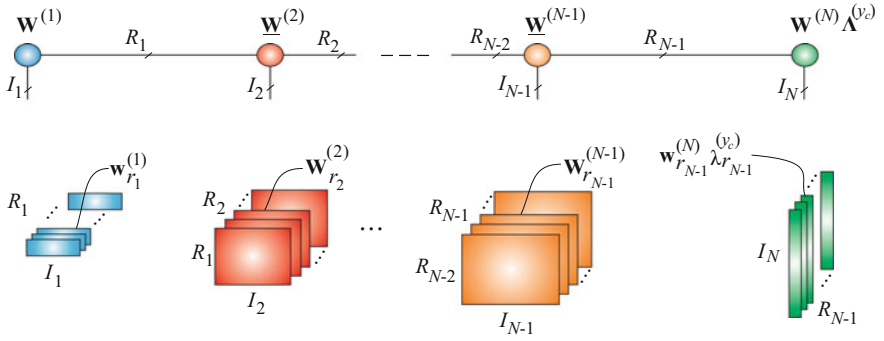

Fig. 14 Basic Tensor Train (TT/MPS) architecture for the representation of coefficient (weight) tensors $\underline{\mathbf{W}}_{y_c}$ of the set of score functions h_{y_c}

The Tensor Train (TT) format can be interpreted as a special case of the HT format, where all nodes (TT-cores) of the underlying tensor network are connected in cascade (or train), i.e., they are aligned while factor matrices corresponding to the leaf modes are assumed to be identities and thus need not be stored. The TT format was first proposed in numerical analysis and scientific computing by Oseledets [15, 69].

Figure 14 presents the concept of TT decomposition for an Nth-order tensor, the entries of which can be computed as a cascaded (multilayer) multiplication of appropriate matrices (slices of TT-cores). The weights of internal edges (denoted by $\{R_1, R_2, \ldots, R_{N-1}\}$) represent the TT-rank. In this way, the so aligned sequence of core tensors represents a "tensor train" where the role of "buffers" is played by TT-core connections. It is important to highlight that TT networks can be applied not only for the approximation of tensorized vectors but also for scalar multivariate functions, matrices, and even large-scale low-order tensors [13].

In the quantum physics community, the TT format is known as the Matrix Product State (MPS) representation with the Open Boundary Conditions (OBC). In fact, the TT/MPS was rediscovered several times under different names: MPS, valence bond states, and density matrix renormalization group (DMRG) (see [13, 27, 89–94] and references therein).

An important advantage of the TT/MPS format over the HT format is its simpler practical implementation, as no binary tree needs to be determined. Another attractive property of the TT-decomposition is its simplicity when performing basic mathematical operations on tensors directly in the TT-format (that is, employing only core tensors). These include matrix-by-matrix and matrix-by-vector multiplications, tensor addition, and the entry-wise (Hadamard) product of tensors. These operations produce tensors, also in the TT-format, which generally exhibit increased TT-ranks. A detailed description of basic operations supported by the TT format is given in [13]. Note that only TT-cores need to be stored and processed, which makes the number of TN parameters to scale linearly in the tensor order, N, of a data

tensor and all mathematical operations are then performed only on the low-order and relatively small size core tensors.

The TT rank is defined as an $(N-1)$-tuple of the form

$$\text{rank}_{\text{TT}}(\underline{\mathbf{X}}) = \mathbf{r}_{TT} = \{R_1, \ldots, R_{N-1}\}, \quad R_n = \text{rank}(\mathbf{X}_{<n>}), \tag{40}$$

where $\mathbf{X}_{<n>} \in \mathbb{R}^{I_1 \cdots I_n \times I_{n-1} \cdots I_N}$ is an nth canonical matricization of the tensor $\underline{\mathbf{X}}$. Since the TT rank determines memory requirements of a tensor train, it has a strong impact on the complexity, i.e., the suitability of tensor train representation for a given raw data tensor.

The number of data samples to be stored scales linearly in the tensor order, N, and the size, I, and quadratically in the maximum TT rank bound, R, that is

$$\sum_{n=1}^{N} R_{n-1} R_n I_n \sim \mathcal{O}(NR^2 I), \quad R := \max_n\{R_n\}, \quad I := \max_n\{I_n\}. \tag{41}$$

This is why it is crucially important to have low-rank TT approximations.[8]

As illustrated in Fig. 14 the simplest possible implementation of the ConvAC network is via the standard tensor train (TT/MPS) (unbalanced binary tree), which can be represented by recursive formulas as

$$\begin{aligned}
\underline{\mathbf{W}}^{\leq 1} &= \mathbf{W}^{(1)} \\
\underline{\mathbf{W}}^{\leq 2} &= \sum_{r_1=1}^{R_1} \underline{\mathbf{W}}^{(1)}_{r_1} \circ \mathbf{W}^{(2)}_{r_1} \in \mathbb{R}^{I_1 \times I_2 \times R_2} \\
&\cdots \\
\underline{\mathbf{W}}^{\leq n} &= \sum_{r_{n-1}=1}^{R_{n-1}} \underline{\mathbf{W}}^{\leq n-1}_{r_{n-1}} \circ \mathbf{W}^{(n)}_{r_{n-1}} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n} \\
&\cdots \\
\underline{\mathbf{W}}^{\leq N-1} &= \sum_{r_{N-2}=1}^{R_{N-2}} \underline{\mathbf{W}}^{\leq N-2}_{r_{N-2}} \circ \mathbf{W}^{(N-1)}_{r_{N-2}} \in \mathbb{R}^{I_1 \times \cdots \times I_{N-1} \times R_{N-1}} \\
\underline{\mathbf{W}}_{y_c} &= \underline{\mathbf{W}}^{\leq N} = \sum_{r_{N-1}=1}^{R_{N-1}} \lambda^{(y_c)}_{r_{N-1}} (\underline{\mathbf{W}}^{\leq N-1}_{r_{N-1}} \circ \mathbf{w}^{(N)}_{r_{N-1},1}) \in \mathbb{R}^{I_1 \times \cdots \times I_N},
\end{aligned} \tag{42}$$

where $\mathbf{W}^{(n)}_{r_{n-1}} = \underline{\mathbf{W}}^{(n)}(r_{n-1}, :, :) \in \mathbb{R}^{I_n \times R_n}$ are lateral slices of the core tensor $\underline{\mathbf{W}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ and $\underline{\mathbf{W}}^{\leq n}_{r_n} = \underline{\mathbf{W}}^{\leq n}(:, \ldots, :, r_n) \in \mathbb{R}^{I_1 \times \cdots \times I_n}$ are sub-tensors of $\underline{\mathbf{W}}^{\leq n} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times R_n}$ for $n = 1, \ldots, N$ (Fig. 14).

The above recursive formulas for the TT network can be written in a compact form as

[8] In the worst case scenario the TT ranks can grow up to $I^{(N/2)}$ for an Nth-order tensor.

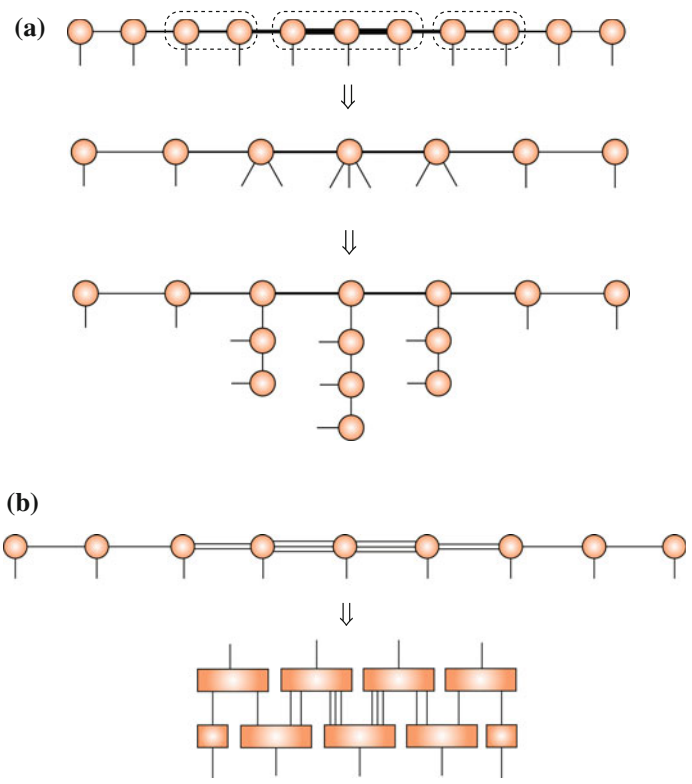

Fig. 15 Extended (modified) Tensor Train (TT/MPS) architectures for the representation of coefficient (weight) tensors $\underline{\mathbf{W}}_{y_c}$ of the score function h_{y_c}. **a** TT-tucker network, also called fork tensor product states (FTPS) with reduced TT ranks. **b** Hierarchical TT network consisting of core tensors with different orders, where each high-order core tensor can be represented by a TT or HT sub-network. Rectangular boxes represent core tensors (sub-tensors) with variable orders. The ticker horizontal lines or double/tripple lines indicate relatively higher internal TT-ranks

$$\underline{\mathbf{W}}_{y_c} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \lambda^{(y_c)}_{r_{N-1}} (\mathbf{w}^{(1)}_{1,r_1} \circ \mathbf{w}^{(2)}_{r_1,r_2} \circ \cdots \circ \mathbf{w}^{(N-1)}_{r_{N-2},r_{N-1}} \circ \mathbf{w}^{(N)}_{r_{N-1},1}), \tag{43}$$

where $\mathbf{w}^{(n)}_{r_{n-1},r_n} = \underline{\mathbf{W}}^{(n)}(:, i_n, :) \in \mathbb{R}^{I_n}$ are tubes of the core tensor $\underline{\mathbf{W}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$.

The TT-rank of the standard tensor train network with 3rd-order cores, shown in Fig. 15, can be very large for core tensors located in the middle of the chain. In the worst case scenario, the TT ranks can grow even, up to $I^{(N/2)}$ for an exact representation of Nth-order tensor. In order to reduce the TT-rank and consequently reduce the number of parameters of a DCNN, we can apply two approaches, as explained in Fig. 14a and b. The main idea is to employ only a few core tensors with order larger than three, whereby each of these core is further approximated via a TT or HT sub-network.

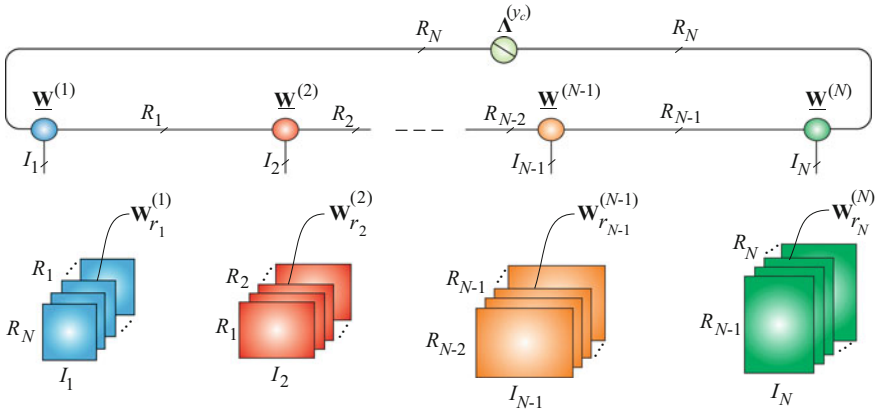

Fig. 16 The Tensor Chain (TC) architecture for the representation of coefficient (weight) tensors $\underline{\mathbf{W}}_{y_c}$ of the score function $h_{y_c}(\mathbf{x}_1, \ldots, \mathbf{x}_N)$. Note that for $R_0 = R_N = 1$ the TC simplifies to the standard TT

6.3 Tensor Chain and TT/MPO Networks

Alternatively, the Tensor Chain (TC) network, called also TT/MPS with periodic bounded conditions, shown in Fig. 16 can be employed to represent individual hidden layers or output fully connected layers. This TC network is mathematically described through the following a recursive formulas as

$$
\begin{aligned}
\underline{\mathbf{W}}^{\leq 1} &= \underline{\mathbf{W}}^{(1)} \in \mathbb{R}^{R_N \times I_1 \times R_1} \\
\underline{\mathbf{W}}^{\leq 2} &= \sum_{r_1=1}^{R_1} \underline{\mathbf{W}}^{\leq 1}_{r_1} \circ \mathbf{W}^{(2)}_{r_1} \in \mathbb{R}^{R_N \times I_1 \times I_2 \times R_2} \\
&\cdots \\
\underline{\mathbf{W}}^{\leq n} &= \sum_{r_{n-1}=1}^{R_{n-1}} \underline{\mathbf{W}}^{\leq n-1}_{r_{n-1}} \circ \mathbf{W}^{(n)}_{r_{n-1}} \in \mathbb{R}^{R_N \times I_1 \times \cdots \times I_n \times R_n} \\
&\cdots \\
\underline{\mathbf{W}}^{\leq N} &= \sum_{r_{N-1}=1}^{R_{N-1}} \underline{\mathbf{W}}^{\leq N-1}_{r_{N-1}} \circ \mathbf{W}^{(N)}_{r_{N-1}} \in \mathbb{R}^{R_N \times I_1 \times \cdots \times I_N \times R_N} \\
\underline{\mathbf{W}}_{y_c} &= \sum_{r_N=1}^{R_N} \lambda^{(y_c)}_{r_N} \underline{\mathbf{W}}^{\leq N}_{r_N, r_N} \in \mathbb{R}^{I_1 \times \cdots \times I_N},
\end{aligned}
\tag{44}
$$

where $\underline{\mathbf{W}}^{(n)}_{r_{n-1}} = \underline{\mathbf{W}}^{(n)}(r_{n-1}, :, :) \in \mathbb{R}^{I_n \times R_n}$ are lateral slices of the core tensor $\underline{\mathbf{W}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$, $\underline{\mathbf{W}}^{\leq n}_{r_n} = \underline{\mathbf{W}}^{\leq n}(:, \ldots, :, r_n) \in \mathbb{R}^{R_N \times I_1 \times \cdots \times I_n}$ are sub-tensors of $\underline{\mathbf{W}}^{\leq n} \in$

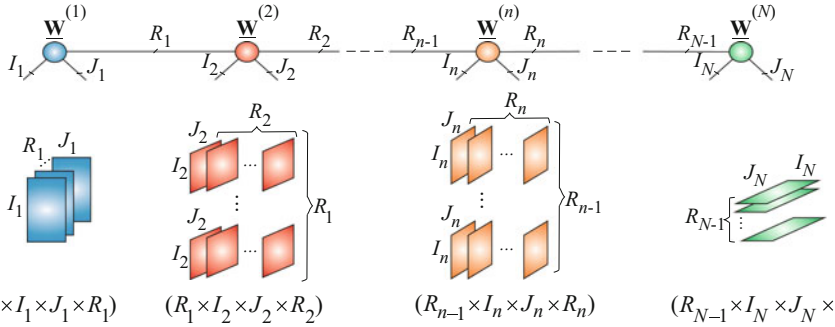

Fig. 17 An alternative TT/MPO tensor network for the approximation of $2N$th-order coefficient tensor $\underline{\mathbf{W}}_{y_c}$ of the score function $h_c(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ defined by Eq. (46). Operation between matrices (slices) of core tensors can be performed multi-linearly and in nonlinear way, as explained in the next section

$\mathbb{R}^{R_N \times I_1 \times \cdots \times I_n \times R_n}$, and $\underline{\mathbf{W}}^{\leq N}_{r_N, r_N} = \underline{\mathbf{W}}^{\leq N}(r_N, :, \ldots, :, r_N) = \text{Tr}(\underline{\mathbf{W}}^{\leq N}) \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ for $n = 1, \ldots, N$ and $R_0 = R_N \neq 1$.

The above TC network can be written in a compact form (due to the commutativity and associativity of the outer products) as

$$\underline{\mathbf{W}}_{y_c} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_N=1}^{R_N} \lambda^{(y_c)}_{r_N} (\mathbf{w}^{(1)}_{r_N, r_1} \circ \mathbf{w}^{(2)}_{r_1, r_2} \circ \cdots \circ \mathbf{w}^{(N)}_{r_{N-1}, r_N}), \tag{45}$$

where $\mathbf{w}^{(n)}_{r_{n-1}, r_n} = \underline{\mathbf{W}}^{(n)}(:, i_n, :) \in \mathbb{R}^{I_n}$ are tubes of a core tensor $\underline{\mathbf{W}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$.

The ConvAC can be alternatively modeled via TT/MPO networks, as illustrated in Fig. 17 for a more general score function defined as

$$h_{y_c} = \sum_{i_1=1}^{I_1} \cdots \sum_{i_N=1}^{I_N} \sum_{j_1=1}^{J_1} \cdots \sum_{j_N=1}^{J_N} \underline{\mathbf{W}}_{y_c}(i_1, \ldots, i_N, j_1, \ldots, j_N) \prod_{n=1}^{N} f_{\theta_{i_n j_n}}(\mathbf{x}_n), \tag{46}$$

where $\underline{\mathbf{W}}_{y_c}(i_1, \ldots, i_N, j_1, \ldots, j_N)$ represent entries of an $2N$th-order coefficient tensor.

Such tensor networks are well understood and efficient algorithms exist to perform their learning, that is, to estimate the core tensors on the basis of a relatively small number of measurements or a small set of available training data.

7 Deep Convolutional Rectifier Using Nonlinear Tensor Networks Decompositions

The convolutional arithmetic circuits (ConvACs) model employs the standard outer (tensor) products, which for two tensors, $\underline{\mathbf{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ and $\underline{\mathbf{B}} \in \mathbb{R}^{J_1 \times \cdots \times J_M}$, are defined as

$$(\underline{\mathbf{A}} \circ \underline{\mathbf{B}})_{i_1,\ldots,i_N j_1,\ldots,j_M} = a_{i_1,\ldots,i_N} \cdot b_{j_1,\ldots,j_M}.$$

However, in order to convert ConvAC tensor models to popular and widely used convolutional rectifier networks we need to employ the generalized (nonlinear) outer products, defined as [30]

$$(\underline{\mathbf{A}} \circ_{\rho} \underline{\mathbf{B}})_{i_1,\ldots,i_N j_1,\ldots,j_M} = \rho(a_{i_1,\ldots,i_N}, b_{j_1,\ldots,j_M}), \tag{47}$$

where the operator

$$\rho = \rho_{\sigma,P}(a, b) = P[\sigma(a), \sigma(b))], \tag{48}$$

is referred to as the activation-pooling operator or function,[9] which meets the associativity and the commutativity requirements (i.e., the operator satisfies the following properties: $\rho(\rho(a, b), c) = \rho(a, \rho(b, c))$ and $\rho(a, b) = \rho(b, a), \quad \forall a, b, c \in \mathbb{R}$).

The activation–pooling operator can take various forms. In particular, for the convolutional rectifier network with max pooling, we can use the following activation-pooling operator

$$\rho_{\sigma,P}(a, b) = \max\{[a]_+, [b]_+)\} = \max\{a, b, 0\}. \tag{49}$$

As an example, consider a generalized CP decomposition, which represents a shallow rectifier network in the form

$$\underline{\mathbf{W}}_{y_c} = \sum_{r=1}^{R} \lambda_r^{(y_c)} \, (\mathbf{w}_r^{(1)} \circ_{\rho} \mathbf{w}_r^{(2)} \circ_{\rho} \cdots \circ_{\rho} \mathbf{w}_r^{(N)}), \tag{50}$$

where the coefficients $\lambda_r^{(y_c)}$ represent weights of the output layer, vectors $\mathbf{w}_r^{(n)} \in \mathbb{R}^{I_n}$ are weights in the hidden layer, and R denotes the number of channels (using the language of deep learning community).

It should be noted that if we employ the weight sharing, then all vectors $\mathbf{w}_r^{(n)} = \mathbf{w}_r$, $\forall n$, and consequently the coefficient tensor, $\underline{\mathbf{W}}_{y_c}$, must be a symmetric tensor which further limits the ability of this model to approximate a desired function.

[9] The symbols $\sigma(\cdot)$ and $P(\cdot)$ are respectively the activation and pooling functions of the network.

As a second example, let us consider a nonlinear HT tensor network which models a deep convolutional rectifier. The TN shown in Fig. 17 can be compactly described as follows (assuming the generalized outer products defined above):

$$\underline{\mathbf{W}}_{y_c} = \sum_{r_1=1}^{R_1} \cdots \sum_{r_{N-1}=1}^{R_{N-1}} \lambda_{r_{N-1}}^{(y_c)} (\mathbf{W}_{1,r_1}^{(1)} \circ_\rho \mathbf{W}_{r_1,r_2}^{(2)} \circ_\rho \cdots \circ_\rho \mathbf{W}_{r_{N-1},1}^{(N-1)}), \tag{51}$$

where $\mathbf{W}_{r_{n-1},r_n}^{(n)} \in \mathbb{R}^{I_n \times J_n}$ are block matrices of core tensor $\underline{\mathbf{W}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ (for more detail see [13, 14]).

The TT and TC networks[10] provide some simplicity in comparison to HT, together with very deep TN structures, that is, N hidden layers. Note that the HT model generates architectures of DCNNs with $L = \log_2(N)$ hidden layers, while TT/TC tenor network employs N hidden layers. Taking into account the current trend in deep leaning to use a large number of hidden layers, it would be a quite attractive to employ so called quantized TT/TC QTT/QTC networks with a relatively large number of hidden layers: $L = N \cdot \log_2(I)$ [13].

To summarize, deep convolutional neural networks may be considered as a special case of hierarchical architectures, which can be indirectly simulated and optimized via relative simple and well understood tensor networks, especially HT/TT (i.e., using unbalanced or balanced binary trees and graphical models), however, more sophisticated tensor network diagrams with loops, discussed in the next section may provide potentially better performance and the ability to generate novel architectures of DCNNs.

8 MERA Tensor Networks for a Next Generation of DCNNs

The Multiscale Entanglement Renormalization Ansatz (MERA) tensor network was first introduced by Vidal [95], and numerical algorithms to minimize the energy or local Hamiltonian already exist [96].

The MERA is a relatively new tensor network, widely investigated in quantum physics based variational Ansatz, since it is capable of capturing many of the key complex physical properties of strongly correlated ground states [97]. The MERA also shares many relationships with the AdS/CFT (gauge-gravity) correspondence by realizing a complete holographic duality within the tensor networks framework. Furthermore, the MERA can be regarded as a TN realization of an orthogonal wavelets transform acting on the mode space of the physical fermionic degrees of freedom [98–100].

[10] It is important to note that TT/TC tensor networks described in this section do not necessary need to have weight sharing and do not need even to be convolutional.

For simplicity, we focus in this section on the 1D binary and ternary MERA tensor networks (see Fig. 18a for basic binary MERA). Instead of writing of complex mathematical formulas it is more convenient to describe MERA tensor networks graphically.

Using, the terminology from quantum physics, the standard binary MERA architecture contains three classes of core tensors: (i) Disentanglers—4th-order cores; (ii) isometries also called the coarse-grainer, typically 3rd order cores for binary MERA and 4th-order cores for ternary MERA; and (iii) one output core which is usually a matrix or a 4th-order core, as illustrated in Fig. 18a and c. Each MERA layer is constructed of a row of disentanglers and a row of coarse-grainers or isometries. Disentanglers remove the short-scale entanglement between the adjacent modes, while isometries renormalise each pair of modes to a single mode Each renormalisation layer performs these operations on a different length scale.

The coarse-grainers take inputs from two modes on a lower scale in the MERA, and give an output onto one mode which is on a higher layer in the tensor network, while the disentangler removes entanglement between two neighboring modes (sites) on the same level. From the perspective of a mapping, the nodes (core tensors) can be considered as processing units, that is, the 4th-order cores map matrices to other matrices, while the coarse-grainers take matrices and map then to vectors. The key idea here is to realize that the "compression" capability arises from the hierarchy and the entanglement. As a matter of fact, the MERA network embodies the mutual information chain rule. In other words, the main idea underlying MERA is that of disentangling the system at various length scales as one follows coarse graining Renormalization Group (RG) flow in the system. The MERA is particularly effective for (scale invariant) critical points of the physical systems.

The key features properties of MERA can be summarized as follows [97]:

- MERA can capture scale-invariance of inputs data;
- It reproduces polynomial decay of correlations between inputs, in contrast to HT or TT tensor networks which reproduce only exponential decay of correlations;
- MERA has ability to much better compress tensor data that TT/HT tensor networks;
- It reproduces a logarithmic correction to the area law, therefore MERA is a more powerful tensor network in comparison to HT/TTNS or TT/TC networks;
- MERA can be efficiently contracted due to unitary constraints imposed on core tensors.

Motivated by these features, we are currently investigating MERA tensor networks as powerful tools to model and analyze DCNNs. A key objective is to establish a precise connection between MERA tensor networks and extended model of DCNNs. This connection may provide exciting new insights about deep learning and may also allow for construction of improved families of DCNNs, with potential application to more efficient data/image classification, clustering and prediction. In other words, we conjecture that the MERA will lead to useful new results, potentially allowing not only better characterization of expressive power of DCNNs, but

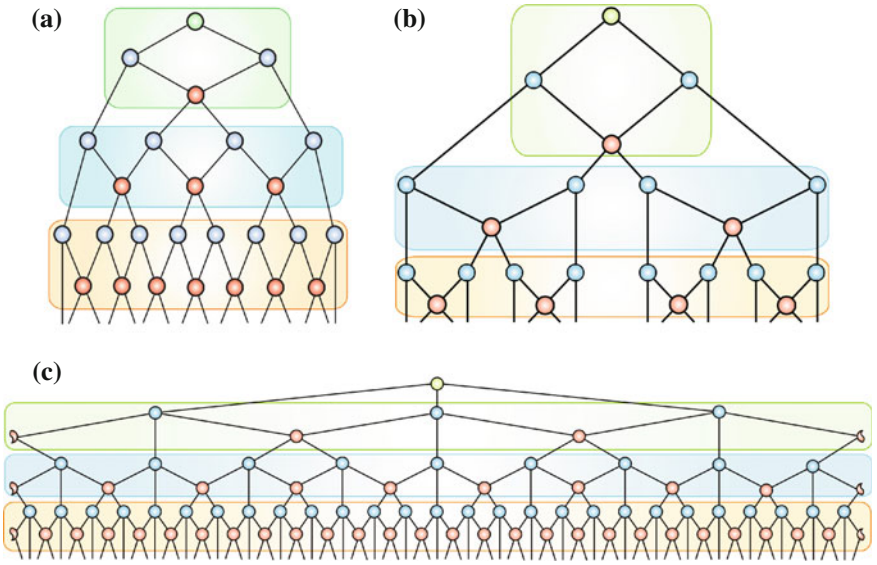

Fig. 18 Various architectures of MERA tensor networks for the new generation of deep convolutional neural networks. **a** Basic binary MERA tensor network. Observe that the alternating layers of disentangling and coarse-graining cores. For the network shown in (**a**) the number of modes (tensor cores) after each such set of operations is approximately halved. **b** Improved (lower complexity) MERA network. **c** Ternary MERA in which coarse grainers are also 4th-order tensors, i.e., three sites (modes) are coarse-grained into one effective site (mode)

also new practical implementations. Going the other way, the links and relations between TNs and DCNNs could lead to useful advances in the design of novel deep neural networks.

The MERA tensor networks, shown in Fig. 18, may provide a much higher expressive power of deep learning in comparison to networks corresponding to HT/TT architectures, since this class of tensor networks can model more complex long term correlations between input instances. This follows form the facts that for HT and TT, TC tensor networks correlations between input variables decay exponentially and the entanglement entropy saturates to a constant, while the more sophisticated MERA tensor networks provide polynomially decaying correlations.

For future research directions, it would be very important to further explore the links between deep learning architectures, such as DCNN or deep Boltzmann machine, and TNs with hierarchical structures such as tree tensor network states (TTNS) and multi-scale entanglement renormalization ansatz (MERA), in order to better understand and improve the expressive power of deep feedforward neural networks. We deeply believe that the insights into the theory of tensor networks and quantum many-body physics can provide better theoretical understanding of deep learning, together with the guidance for optimized DNNs design.

To summarize, the tensor network methodology and architectures discussed in this section could be extended to allow analytic construction of new DCNNs. Moreover, systematic investigation of the correspondences between DNNs and wide spectrum of TNs can provide a very fruitful perspective including cashing the existing conjectures and claims about operational similarities and correspondences between DNNs and TNs into a more rigorous and constructive framework.

9 Conclusions and Discussions

The tensor networks (TNs) methodology is a promising paradigm for the analysis of extreme-scale multidimensional data. Due to their 'super' compression abilities and the distributed way in which they process data, TNs can be employed for a wide family of large-scale optimization problems, especially linear/multilinear dimensionality reduction tasks.

In this paper, we focused on two main challenges in huge-scale data analysis which are addressed by tensor networks: (i) an approximate representation of a specific cost (objective) function by a tensor network while maintaining the desired accuracy of approximation, and (ii) the extraction of physically meaningful latent variables from data in a sufficiently accurate and computationally affordable way. The benefits of multiway (tensor) analysis methods for large-scale data sets then include:

- Graphical representations of tensor networks allow us to express mathematical operations on tensors (e.g., tensor contractions and reshaping) in a simple and intuitive way, and without the explicit use of complex mathematical expressions;
- Simultaneous and flexible distributed representations of both the structurally rich data and complex optimization tasks;
- Efficient compressed formats of large multidimensional data achieved via tensorization and low-rank tensor decompositions into low-order factor matrices and/or core tensors;
- Ability to operate with noisy and missing data by virtue of numerical stability and robustness to noise of low-rank tensor/matrix approximation algorithms;
- A flexible framework which naturally incorporates various diversities and constraints, thus seamlessly extending the standard, flat view, Component Analysis (2-way CA) methods to multiway component analysis;
- Possibility to analyze linked (coupled) blocks of large-scale matrices and tensors in order to separate common/correlated from independent/uncorrelated components in the observed raw data.

In that sense, this paper both reviews current research in this area and complements optimisation methods, such as the Alternating Direction Method of Multipliers (ADMM) [101].

Tensor decompositions (TDs) have been already adopted in widely diverse disciplines, including psychometrics, chemometrics, biometric, quantum physics quantum chemistry, signal and image processing, machine learning, and brain science [12–14, 26, 29, 63, 102–105]. This is largely due to their advantages in the analysis of data that exhibit not only large volume but also very high variety (see Fig. 1), as in the case in bio- and neuroinformatics and in computational neuroscience, where various forms of data collection include sparse tabular structures and graphs or hypergraphs.

Moreover, tensor networks have the ability to efficiently parameterize, through structured compact representations, very general high-dimensional spaces which arise in modern applications [11, 14, 72, 106–110]. Tensor networks also naturally account for intrinsic multidimensional and distributed patterns present in data, and thus provide the opportunity to develop very sophisticated models for capturing multiple interactions and couplings in data—these are more physically insightful and interpretable than standard pair-wise interactions.

Acknowledgements This work has been partially supported by Misnistry of Education and Science of the Russian Federation (grant 14.756,0001).

References

1. Zurada, J.: Introduction to Artificial Neural Systems, vol. 8. West St, Paul (1992)
2. LeCun, Y., Bengio, Y.: Convolutional networks for images, speech, and time series. In: The Handbook of Brain Theory and Neural Networks, MIT Press, pp. 255–258 (1998)
3. Hinton, G., Sejnowski, T.: Learning and relearning in boltzmann machines. In: Parallel Distributed Processing, MIT Press, pp. 282–317 (1986)
4. Cichocki, A., Kasprzak, W., Amari, S.: Multi-layer neural networks with a local adaptive learning rule for blind separation of source signals. In: Proceedings of the International Symposium Nonlinear Theory and Applications (NOLTA), Las Vegas, NV, Citeseer, pp. 61–65 (1995)
5. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
6. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
7. Cichocki, A., Zdunek, R.: Multilayer nonnegative matrix factorisation. Electron. Lett. **42**(16), 1 (2006)
8. Cichocki, A., Zdunek, R.: Regularized alternating least squares algorithms for non-negative matrix/tensor factorization. In: International Symposium on Neural Networks, pp. 793–802. Springer (2007)
9. Cichocki, A.: Tensor decompositions: new concepts in brain data analysis? J. Soc. Instr. Control Eng. **50**(7), 507–516. arXiv:1305.0395 (2011)
10. Cichocki, A.: Era of big data processing: a new approach via tensor networks and tensor decompositions, (invited). In: Proceedings of the International Workshop on Smart Info-Media Systems in Asia (SISA2013). arXiv:1403.2048 (September 2013)
11. Cichocki, A.: Tensor networks for big data analytics and large-scale optimization problems. arXiv:1407.3124 (2014)

12. Cichocki, A., Mandic, D., Caiafa, C., Phan, A., Zhou, G., Zhao, Q., Lathauwer, L.D.: Tensor decompositions for signal processing applications: from two-way to multiway component analysis. IEEE Signal Process. Mag. **32**(2), 145–163 (2015)
13. Cichocki, A., Lee, N., Oseledets, I., Phan, A.H., Zhao, Q., Mandic, D.: Tensor networks for dimensionality reduction and large-scale optimization: part 1 low-rank tensor decompositions. Found. Trends Mach. Learn. **9**(4–5), 249–429 (2016)
14. Cichocki, A., Phan, A.H., Zhao, Q., Lee, N., Oseledets, I., Sugiyama, M., Mandic, D.: Tensor networks for dimensionality reduction and large-scale optimization: part 2 applications and future perspectives. Found. Trends Mach. Learn. **9**(6), 431–673 (2017)
15. Oseledets, I., Tyrtyshnikov, E.: Breaking the curse of dimensionality, or how to use SVD in many dimensions. SIAM J. Sci. Comput. **31**(5), 3744–3759 (2009)
16. Dolgov, S., Khoromskij, B.: Two-level QTT-Tucker format for optimized tensor calculus. SIAM J. Matrix Anal. Appl. **34**(2), 593–623 (2013)
17. Kazeev, V., Khoromskij, B., Tyrtyshnikov, E.: Multilevel Toeplitz matrices generated by tensor-structured vectors and convolution with logarithmic complexity. SIAM J. Sci. Comput. **35**(3), A1511–A1536 (2013)
18. Kazeev, V., Khammash, M., Nip, M., Schwab, C.: Direct solution of the chemical master equation using quantized tensor trains. PLoS Comput. Biol. **10**(3), e1003359 (2014)
19. Kressner, D., Steinlechner, M., Uschmajew, A.: Low-rank tensor methods with subspace correction for symmetric eigenvalue problems. SIAM J. Sci. Comput. **36**(5), A2346–A2368 (2014)
20. Vervliet, N., Debals, O., Sorber, L., De Lathauwer, L.: Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis. IEEE Signal Process. Mag. **31**(5), 71–79 (2014)
21. Dolgov, S., Khoromskij, B.: Simultaneous state-time approximation of the chemical master equation using tensor product formats. Numer. Linear Algebra Appl. **22**(2), 197–219 (2015)
22. Liao, S., Vejchodský, T., Erban, R.: Tensor methods for parameter estimation and bifurcation analysis of stochastic reaction networks. J. R. Soc. Interface **12**(108), 20150233 (2015)
23. Bolten, M., Kahl, K., Sokolović, S.: Multigrid methods for tensor structured Markov chains with low rank approximation. SIAM J. Sci. Comput. **38**(2), A649–A667 (2016)
24. Lee, N., Cichocki, A.: Estimating a few extreme singular values and vectors for large-scale matrices in Tensor Train format. SIAM J. Matrix Anal. Appl. **36**(3), 994–1014 (2015)
25. Lee, N., Cichocki, A.: Regularized computation of approximate pseudoinverse of large matrices using low-rank tensor train decompositions. SIAM J. Matrix Anal. Appl. **37**(2), 598–623 (2016)
26. Kolda, T., Bader, B.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009)
27. Orús, R.: A practical introduction to tensor networks: matrix product states and projected entangled pair states. Ann. Phys. **349**, 117–158 (2014)
28. Dolgov, S., Savostyanov, D.: Alternating minimal energy methods for linear systems in higher dimensions. SIAM J. Sci. Comput. **36**(5), A2248–A2271 (2014)
29. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.: Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation. Wiley, Chichester (2009)
30. Cohen, N., Shashua, A.: Convolutional rectifier networks as generalized tensor decompositions. In: Proceedings of The 33rd International Conference on Machine Learning, pp. 955–963 (2016)
31. Li, J., Battaglino, C., Perros, I., Sun, J., Vuduc, R.: An input-adaptive and in-place approach to dense tensor-times-matrix multiply. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, p. 76. ACM (2015)

32. Ballard, G., Benson, A., Druinsky, A., Lipshitz, B., Schwartz, O.: Improving the numerical stability of fast matrix multiplication algorithms. arXiv:1507.00687 (2015)
33. Ballard, G., Druinsky, A., Knight, N., Schwartz, O.: Brief announcement: Hypergraph partitioning for parallel sparse matrix-matrix multiplication. In: Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, pp. 86–88. ACM (2015)
34. Tucker, L.: The extension of factor analysis to three-dimensional matrices. In: Gulliksen, H., Frederiksen, N. (eds.) Contributions to Mathematical Psychology, pp. 110–127. Holt, Rinehart and Winston, New York (1964)
35. Tucker, L.: Some mathematical notes on three-mode factor analysis. Psychometrika **31**(3), 279–311 (1966)
36. Sun, J., Tao, D., Faloutsos, C.: Beyond streams and graphs: dynamic tensor analysis. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge Discovery and Data Mining, pp. 374–383. ACM (2006)
37. Drineas, P., Mahoney, M.: A randomized algorithm for a tensor-based generalization of the singular value decomposition. Linear Algebra Appl. **420**(2), 553–571 (2007)
38. Lu, H., Plataniotis, K., Venetsanopoulos, A.: A survey of multilinear subspace learning for tensor data. Pattern Recogn. **44**(7), 1540–1551 (2011)
39. Li, M., Monga, V.: Robust video hashing via multilinear subspace projections. IEEE Trans. Image Process. **21**(10), 4397–4409 (2012)
40. Pham, N., Pagh, R.: Fast and scalable polynomial kernels via explicit feature maps. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 239–247. ACM (2013)
41. Wang, Y., Tung, H.Y., Smola, A., Anandkumar, A.: Fast and guaranteed tensor decomposition via sketching. In: Advances in Neural Information Processing Systems, pp. 991–999 (2015)
42. Kuleshov, V., Chaganty, A., Liang, P.: Tensor factorization via matrix factorization. In: Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, pp. 507–516 (2015)
43. Sorber, L., Domanov, I., Van Barel, M., De Lathauwer, L.: Exact line and plane search for tensor optimization. Comput. Optim. Appl. **63**(1), 121–142 (2016)
44. Lubasch, M., Cirac, J., Banuls, M.C.: Unifying projected entangled pair state contractions. New J. Phys. **16**(3), 033014 (2014)
45. Di Napoli, E., Fabregat-Traver, D., Quintana-Ortí, G., Bientinesi, P.: Towards an efficient use of the BLAS library for multilinear tensor contractions. Appl. Math. Comput. **235**, 454–468 (2014)
46. Pfeifer, R., Evenbly, G., Singh, S., Vidal, G.: NCON: A tensor network contractor for MATLAB. arXiv:1402.0939 (2014)
47. Kao, Y.J., Hsieh, Y.D., Chen, P.: Uni10: An open-source library for tensor network algorithms. J. Phys. Conf. Ser. **640**, 012040 (2015). IOP Publishing
48. Grasedyck, L., Kessner, D., Tobler, C.: A literature survey of low-rank tensor approximation techniques. GAMM-Mitteilungen **36**, 53–78 (2013)
49. Comon, P.: Tensors: A brief introduction. IEEE Signal Process. Mag. **31**(3), 44–53 (2014)
50. Sidiropoulos, N., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E., Faloutsos, C.: Tensor decomposition for signal processing and machine learning. arXiv:1607.01668 (2016)
51. Zhou, G., Cichocki, A.: Fast and unique Tucker decompositions via multiway blind source separation. Bull. Pol. Acad. Sci. **60**(3), 389–407 (2012)
52. Phan, A., Cichocki, A., Tichavský, P., Zdunek, R., Lehky, S.: From basis components to complex structural patterns. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26–31, 2013, pp. 3228–3232
53. Phan, A., Tichavský, P., Cichocki, A.: Low rank tensor deconvolution. In: Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP, April 2015, pp. 2169–2173
54. Lee, N., Cichocki, A.: Fundamental tensor operations for large-scale data analysis using tensor network formats. Multidimension. Syst. Signal Process, pp 1–40, Springer (March 2017)

55. Bellman, R.: Adaptive Control Processes. Princeton University Press, Princeton, NJ (1961)
56. Austin, W., Ballard, G., Kolda, T.: Parallel tensor compression for large-scale scientific data. arXiv:1510.06689 (2015)
57. Jeon, I., Papalexakis, E., Faloutsos, C., Sael, L., Kang, U.: Mining billion-scale tensors: algorithms and discoveries. VLDB J. 1–26 (2016)
58. Phan, A., Cichocki, A.: PARAFAC algorithms for large-scale problems. Neurocomputing **74**(11), 1970–1984 (2011)
59. Klus, S., Schütte, C.: Towards tensor-based methods for the numerical approximation of the Perron-Frobenius and Koopman operator. arXiv:1512.06527 (December 2015)
60. Bader, B., Kolda, T.: MATLAB tensor toolbox version. **2**, 6 (2015)
61. Garcke, J., Griebel, M., Thess, M.: Data mining with sparse grids. Computing **67**(3), 225–253 (2001)
62. Bungartz, H.J., Griebel, M.: Sparse grids. Acta Numerica **13**, 147–269 (2004)
63. Hackbusch, W.: Tensor spaces and numerical tensor calculus. Springer Series in Computational Mathematics, vol. 42. Springer, Heidelberg (2012)
64. Bebendorf, M.: Adaptive cross-approximation of multivariate functions. Constr. Approx. **34**(2), 149–179 (2011)
65. Dolgov, S.: Tensor product methods in numerical simulation of high-dimensional dynamical problems. Ph.D. thesis, Faculty of Mathematics and Informatics, University Leipzig, Germany, Leipzig, Germany (2014)
66. Cho, H., Venturi, D., Karniadakis, G.: Numerical methods for high-dimensional probability density function equations. J. Comput. Phys. **305**, 817–837 (2016)
67. Trefethen, L.: Cubature, approximation, and isotropy in the hypercube. SIAM Rev. (to appear) (2017)
68. Oseledets, I., Dolgov, S., Kazeev, V., Savostyanov, D., Lebedeva, O., Zhlobich, P., Mach, T., Song, L.: TT-Toolbox (2012)
69. Oseledets, I.: Tensor-train decomposition. SIAM J. Sci. Comput. **33**(5), 2295–2317 (2011)
70. Khoromskij, B.: Tensors-structured numerical methods in scientific computing: Survey on recent advances. Chemometr. Intell. Lab. Syst. **110**(1), 1–19 (2011)
71. Oseledets, I., Tyrtyshnikov, E.: TT cross-approximation for multidimensional arrays. Linear Algebra Appl. **432**(1), 70–88 (2010)
72. Khoromskij, B., Veit, A.: Efficient computation of highly oscillatory integrals by using QTT tensor approximation. Comput. Methods Appl. Math. **16**(1), 145–159 (2016)
73. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
74. Schneider, D.: Deeper and cheaper machine learning [top tech 2017]. IEEE Spectr. **54**(1), 42–43 (2017)
75. Lebedev, V., Lempitsky, V.: Fast convolutional neural networks using group-wise brain damage. arXiv:1506.02515 (2015)
76. Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D.: Tensorizing neural networks. In: Advances in Neural Information Processing Systems (NIPS), pp. 442–450 (2015)
77. Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., Liao, Q.: Why and when can deep–but not shallow–networks avoid the curse of dimensionality: a review. arXiv:1611.00740 (2016)
78. Yang, Y., Hospedales, T.: Deep multi-task representation learning: a tensor factorisation approach. arXiv:1605.06391 (2016)
79. Cohen, N., Sharir, O., Shashua, A.: On the expressive power of deep learning: a tensor analysis. In: 29th Annual Conference on Learning Theory, pp. 698–728 (2016)
80. Chen, J., Cheng, S., Xie, H., Wang, L., Xiang, T.: On the equivalence of restricted Boltzmann machines and tensor network states. arXiv e-prints (2017)
81. Cohen, N., Shashua, A.: Inductive bias of deep convolutional networks through pooling geometry. CoRR (2016). arXiv:1605.06743
82. Sharir, O., Tamari, R., Cohen, N., Shashua, A.: Tensorial mixture models. CoRR (2016). arXiv:1610.04167

83. Lin, H.W., Tegmark, M.: Why does deep and cheap learning work so well? arXiv e-prints (2016)
84. Zwanziger, D.: Fundamental modular region, Boltzmann factor and area law in lattice theory. Nucl. Phys. B **412**(3), 657–730 (1994)
85. Eisert, J., Cramer, M., Plenio, M.: Colloquium: Area laws for the entanglement entropy. Rev. Modern Phys. **82**(1), 277 (2010)
86. Calabrese, P., Cardy, J.: Entanglement entropy and quantum field theory. J. Stat. Mech. Theory Exp. **2004**(06), P06002 (2004)
87. Anselmi, F., Rosasco, L., Tan, C., Poggio, T.: Deep convolutional networks are hierarchical kernel machines. arXiv:1508.01084 (2015)
88. Mhaskar, H., Poggio, T.: Deep vs. shallow networks: an approximation theory perspective. Anal. Appl. **14**(06), 829–848 (2016)
89. White, S.: Density-matrix algorithms for quantum renormalization groups. Phys. Rev. B **48**(14), 10345 (1993)
90. Vidal, G.: Efficient classical simulation of slightly entangled quantum computations. Phys. Rev. Lett. **91**(14), 147902 (2003)
91. Perez-Garcia, D., Verstraete, F., Wolf, M., Cirac, J.: Matrix product state representations. Quantum Inf. Comput. **7**(5), 401–430 (2007)
92. Verstraete, F., Murg, V., Cirac, I.: Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems. Adv. Phys. **57**(2), 143–224 (2008)
93. Schollwöck, U.: Matrix product state algorithms: DMRG, TEBD and relatives. In: Strongly Correlated Systems, pp. 67–98. Springer (2013)
94. Huckle, T., Waldherr, K., Schulte-Herbriggen, T.: Computations in quantum tensor networks. Linear Algebra Appl. **438**(2), 750–781 (2013)
95. Vidal, G.: Class of quantum many-body states that can be efficiently simulated. Phys. Rev. Lett. **101**(11), 110501 (2008)
96. Evenbly, G., Vidal, G.: Algorithms for entanglement renormalization. Phys. Rev. B **79**(14), 144108 (2009)
97. Evenbly, G., Vidal, G.: Tensor network renormalization yields the multiscale entanglement renormalization Ansatz. Phys. Rev. Lett. **115**(20), 200401 (2015)
98. Evenbly, G., White, S.R.: Entanglement renormalization and wavelets. Phys. Rev. Lett. **116**(14), 140403 (2016)
99. Evenbly, G., White, S.R.: Representation and design of wavelets using unitary circuits. arXiv e-prints (2016)
100. Matsueda, H.: Analytic optimization of a MERA network and its relevance to quantum integrability and wavelet. arXiv:1608.02205 (2016)
101. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
102. Smilde, A., Bro, R., Geladi, P.: Multi-way Analysis: Applications in the Chemical Sciences. Wiley, New York (2004)
103. Tao, D., Li, X., Wu, X., Maybank, S.: General tensor discriminant analysis and Gabor features for gait recognition. IEEE Trans. Pattern Anal. Mach. Intell. **29**(10), 1700–1715 (2007)
104. Kroonenberg, P.: Applied Multiway Data Analysis. Wiley, New York (2008)
105. Favier, G., de Almeida, A.: Overview of constrained PARAFAC models. EURASIP J. Adv. Signal Process. **2014**(1), 1–25 (2014)
106. Kressner, D., Steinlechner, M., Vandereycken, B.: Low-rank tensor completion by Riemannian optimization. BIT Numer. Math. **54**(2), 447–468 (2014)
107. Zhang, Z., Yang, X., Oseledets, I., Karniadakis, G., Daniel, L.: Enabling high-dimensional hierarchical uncertainty quantification by ANOVA and tensor-train decomposition. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **34**(1), 63–76 (2015)
108. Corona, E., Rahimian, A., Zorin, D.: A tensor-train accelerated solver for integral equations in complex geometries. arXiv:1511.06029 (2015)

109. Litsarev, M., Oseledets, I.: A low-rank approach to the computation of path integrals. J. Comput. Phys. **305**, 557–574 (2016)
110. Benner, P., Khoromskaia, V., Khoromskij, B.: A reduced basis approach for calculation of the Bethe-Salpeter excitation energies by using low-rank tensor factorisations. Mol. Phys. **114**(7–8), 1148–1161 (2016)