Matrix Product Operator Restricted Boltzmann Machines

Chen, Cong; Batselier, Kim; Ko, Ching Yun; Wong, Ngai

DOI
10.1109/IJCNN.2019.8851877

Publication date
2019

Document Version
Final published version

Published in
Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN 2019)

Matrix Product Operator Restricted Boltzmann Machines

Cong Chen*, Kim Batselier**, Ching-Yun Ko*, and Ngai Wong*

*Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China
Email: {chencong, cyko, nwong}@eee.hku.hk
**Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands
Email: k.batselier@tudelft.nl

***Abstract*—A restricted Boltzmann machine (RBM) learns a probability distribution over its input samples and has numerous uses like dimensionality reduction, classification and generative modeling. Conventional RBMs accept vectorized data that dismiss potentially important structural information in the original tensor (multi-way) input. Matrix-variate and tensor-variate RBMs, named MvRBM and TvRBM, have been proposed but are all restrictive by model construction and have weak model expression power. This work presents the matrix product operator RBM (MPORBM) that utilizes a tensor network generalization of Mv/TvRBM, preserves input formats in both the visible and hidden layers, and results in higher expressive power. A novel training algorithm integrating contrastive divergence and an alternating optimization procedure is also developed. Numerical experiments compare the MPORBM with the traditional RBM and MvRBM for data classification and image completion and denoising tasks. The expressive power of the MPORBM as a function of the MPO-rank is also investigated.**

***Index Terms*—tensors, matrix product operators, restricted Boltzmann machines**

I. Introduction

A restricted Boltzmann machine (RBM) [1] is a probabilistic model that employs a layer of hidden variables to achieve highly expressive marginal distributions. RBMs are an unsupervised learning technique and have been extensively explored and applied in various fields, such as pattern recognition [2], computer vision [3] and signal processing [4]. However, conventional RBMs are designed for vector data and cannot directly deal with matrices and higher-dimensional data, which are common in real-life. For example, grayscale images are second-order tensors (i.e. matrices) while color images or grayscale videos are third-order tensors. The traditional approach to apply an RBM on such high-dimensional data is through vectorization of the data, which leads to three drawbacks. First, the spatial information in the original data is lost, thus weakening the model's capability to represent these structural correlations. Second, the fully connected visible and hidden units may cause overfitting since the intrinsic low-rank property of many real-life data is disregarded. Third, in order to train the dense matrix between the visible and hidden layers, much storage and computation resources are required when the layer sizes are large. That may lead to failure in training the RBM.

To address these, we propose a matrix product operator (MPO) restricted Boltzmann machine (MPORBM) wherein both the visible and hidden layers are in tensor forms. The reason we assume the hidden layer is in tensor format is that we can readily use an MPO to connect the visible and hidden layers. An MPO is essentially a tensor network reformulation of the dense matrix in a traditional RBM. In practical cases where the tensor network ranks are low, the number of model parameters can be drastically reduced without affecting the expressive power of the model. To train an MPORBM, we further describe its customized parameter learning algorithms. The MPORBM is also compared with the standard RBM and matrix-variate RBM [5] for tensor inputs in numerical experiments. This article has the following major contributions:

1) The MPORBM is proposed for the first time and generalizes existing RBM architectures. The standard RBM, matrix-variate RBM (MvRBM) [5] and tensor-variate RBM (TvRBM) [6] models are all special cases of the MPORBM.
2) Compared with standard RBMs, the number of parameters in an MPORBM grows only linearly with the order of the tensor instead of exponentially. This alleviates overfitting, which is demonstrated through numerical experiments.
3) Both the visible and hidden layers are represented in tensor forms and therefore useful structural information in the original tensor data is preserved and utilized.
4) The graphical "language" of tensor network diagrams [7] is introduced to represent how specific quantities (such as the conditional probabilities) are computed. This way, complicated mathematical expressions are easily understood in a visual manner.
5) Although the data structures are generalized to tensors, the bipartite graph nature of an RBM still applies, together with the fast sampling and inference properties.

II. Related work

Real-life data are extensively multiway. Researchers have been motivated to develop corresponding multiway RBMs. For example, [8] proposed a factored conditional RBM for modeling motion style. In their model, both historical and

This work is supported by the Hong Kong Research Grants Council under General Research Fund (GRF) Project 17246416, and the University Research Committee of The University of Hong Kong.

978-1-7281-2009-6/$31.00 paper N-20160.pdf

current motion vectors are considered as inputs so that the pairwise association between them is captured. However, since the visible layer is in vector form, the spatial information in the multiway data is not retained. In [3] a three-way factored conditional RBM was proposed where a three-way weight tensor is employed to capture the correlations between the input, output, and hidden variables. However, their training data still require vectorization.

The above works are both aiming to capture the interaction among different vector inputs and are hence not directly applicable to matrix and tensor data. The first RBM designed for tensor inputs is [6], which is called a tensor-variate RBM (TvRBM). In TvRBM, the visible layer is represented as a tensor but the hidden layer is still a vector, which may still suffer from the curse of dimensionality when the hidden layer size is large. Furthermore, the connection between the visible and hidden layers is described by a canonical polyadic (CP) tensor decomposition [9]. However, this CP weight tensor is known to constrain the model representation capability [5].

Another RBM-related model that utilizes tensor input is the matrix-variate RBM (MvRBM) [5]. The visible and hidden layers in an MvRBM are both matrices. Nonetheless, to limit the number of parameters, an MvRBM models the connection between the visible and hidden layers through two separate matrices, which restricts the ability of the model to capture correlations between different data modes.

All these issues have motivated the MPORBM. Specifically, MPORBM not only employs tensorial visible and hidden layers, but also utilizes a general and powerful tensor network, namely an MPO, to connect the tensorial visible and hidden layers. By doing so, an MPORBM achieves a more powerful model representation capacity than MvRBM and at the same time greatly reduces the model parameters compared to a standard RBM. Note that a mapping of the standard RBM with tensor networks has been described in [10]. However, their work does not generalize the standard RBM to tensorial inputs and is therefore still based on visible and hidden units in vector forms.

III. Preliminaries

We review some necessary tensor basics, the MPO tensor decomposition, the standard RBM and its tensorial variants.

A. Tensor basics

Tensors are multi-dimensional arrays that are higher-order generalization of vectors (first-order tensors) and matrices (second-order tensors). A dth-order (also called d-way or d-mode) tensor is denoted as $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$ where each entry $\boldsymbol{\mathcal{A}}(i_1, i_2, \ldots, i_d)$ is determined by d indices $1 \leq i_k \leq I_k (k = 1, 2, \ldots, d)$. The numbers $I_1, I_2, \ldots, I_d$ are called the dimensions of the tensor. We use boldface capital calligraphic letters $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$, ... to denote tensors, boldface capital letters $\boldsymbol{A}$, $\boldsymbol{B}$, ... to denote matrices, boldface letters $\boldsymbol{a}$, $\boldsymbol{b}$, ... to denote vectors, and roman letters a, b, ... to denote scalars. $\boldsymbol{A}^T$ and $\boldsymbol{a}^T$ are the transposes of a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{a}$. An intuitive and useful graphical representation, named tensor network diagrams, of scalars, vectors, matrices and tensors is depicted in Fig. 1. The unconnected edges are the indices of the tensor. We will mainly employ these graphical representations to visualize the tensor networks and operations in the following sections and refer to [7] for more details. We now briefly introduce some important tensor operations.

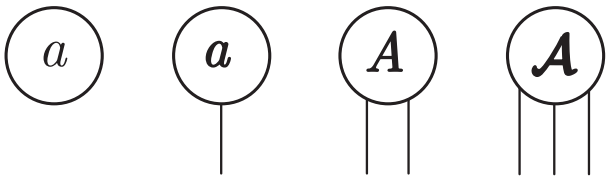

Fig. 1: Graphical representation of a scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$, and third-order tensor $\boldsymbol{\mathcal{A}}$.

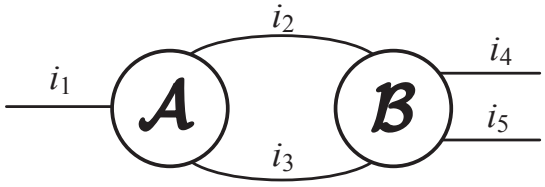

Fig. 2: Tensor contraction between a third-order tensor $\boldsymbol{\mathcal{A}}$ and a fourth-order tensor $\boldsymbol{\mathcal{B}}$, where the summation runs over i_2 and i_3.

Definition 1: (Tensor index contraction): A tensor index contraction is the sum over all possible values of the repeated indices in a set of tensors.

For example, the following contraction between a third-order tensor $\boldsymbol{\mathcal{A}}$ and a fourth-order tensor $\boldsymbol{\mathcal{B}}$

$$\boldsymbol{\mathcal{C}}(i_1, i_4, i_5) = \sum_{i_2=1}^{I_2} \sum_{i_3=1}^{I_3} \boldsymbol{\mathcal{A}}(i_1, i_2, i_3)\, \boldsymbol{\mathcal{B}}(i_2, i_3, i_4, i_5),$$

over the i_2 and i_3 indices results in a third-order tensor $\boldsymbol{\mathcal{C}}$. The corresponding tensor network diagram for these index contractions is shown in Fig. 2, where the summation over the i_2 and i_3 indices is indicated by the connected edges. The resulting diagram has three unconnected indices (i_1, i_4, i_5), which confirms that $\boldsymbol{\mathcal{C}}$ is of third order.

B. Tensor MPO decomposition

An MPO is essentially a tensor network representation of a matrix. To relate the row and column indices of a matrix to the corresponding multi-indices of the MPO we need the following definition.

Definition 2: The mapping between a linear index i of a vector $\boldsymbol{a} \in \mathbb{R}^{I_1 \cdots I_d}$ and its corresponding multi-index $[i_1, i_2, \ldots, i_d]$ when reshaped into a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is

$$i = i_1 + \sum_{k=2}^{d} (i_k - 1) \prod_{p=1}^{k-1} I_p. \tag{1}$$

Now suppose we have a matrix $\boldsymbol{A} \in \mathbb{R}^{I_1 \cdots I_d \times J_1 \cdots J_d}$, where the index mapping (1) is used to split both the row index i and column index j into multi-indices $[i_1, \ldots, i_d], [j_1, \ldots, j_d]$,

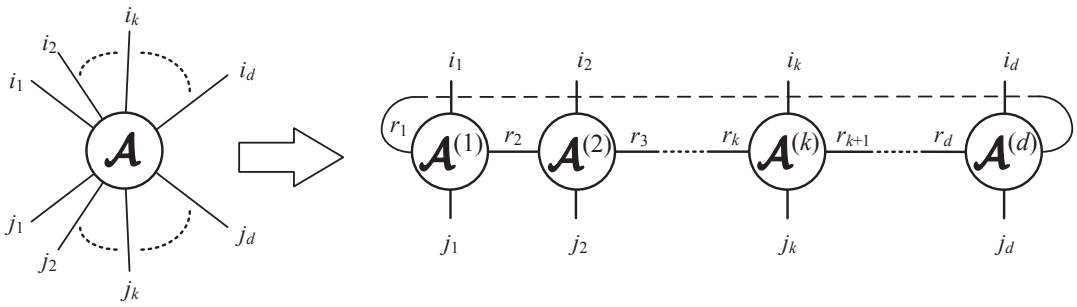

Fig. 3: Matrix product operator (MPO) decomposition.

respectively. After this index splitting we can arrange all matrix entries into a $2d$-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times \cdots \times I_d \times J_1 \times \cdots \times J_d}$. Per definition, the corresponding MPO decomposition represents each entry of $\boldsymbol{\mathcal{A}}$ as

$$\boldsymbol{\mathcal{A}}(i_1, \ldots, i_d, j_1, \ldots, j_d) = \sum_{\substack{R_1, R_2, \ldots, R_d \\ r_1, r_2, \ldots, r_d}} \boldsymbol{\mathcal{A}}^{(1)}(r_1, i_1, j_1, r_2) \cdots \boldsymbol{\mathcal{A}}^{(d)}(r_d, i_d, j_d, r_1). \tag{2}$$

The "building blocks" of the MPO are the 4-way tensors $\boldsymbol{\mathcal{A}}^{(1)}, \ldots, \boldsymbol{\mathcal{A}}^{(d)}$, also called the MPO-cores. The dimensions $R_1, R_2, \ldots, R_d$ of the summation indices $r_1, \ldots, r_d$ are called the MPO-ranks. Observe that the second summation index of $\boldsymbol{\mathcal{A}}^{(d)}$ is the same as the first index of $\boldsymbol{\mathcal{A}}^{(1)}$, which ensures that the right-hand side of (2) results in a scalar. When $R_1 > 1$, the MPO is said to have periodic boundary conditions. We will assume throughout the remainder of this article that $R_1 = 1$. The tensor network diagram representation of (2) is shown in Fig. 3. All index contractions are again represented as edges connecting two tensors. The main motivation to use an MPO representation of a matrix is to reduce its storage requirement. Indeed, using an MPO-representation with maximal MPO-rank R for a matrix $\boldsymbol{A} \in \mathbb{R}^{I^d \times I^d}$ reduces the storage requirement from I^{2d} down to approximately dI^2R^2. This leads to significant storage savings when R is relatively small. Consequently, all computations are done on the MPO-cores rather than on the whole matrix itself. This could potentially reduce the computational complexity of the learning algorithm, especially if the visible and hidden layers are also represented as matrix product states (MPS, also called tensor trains in the scientific computing community [11]). This extension is kept for future work.

C. Standard RBM

The standard RBM [1] is a bipartite undirected probabilistic graphical model with one visible layer $\boldsymbol{v} \in \mathbb{R}^M$ and one hidden layer $\boldsymbol{h} \in \mathbb{R}^N$, both in vector forms. Here we mainly consider binary RBMs, which implies that entries in both $\boldsymbol{v}$ and $\boldsymbol{h}$ attain binary values. The standard RBM assigns the following energy function for a specific joint configuration $\{\boldsymbol{v}, \boldsymbol{h}\}$:

$$E(\boldsymbol{v}, \boldsymbol{h}; \Theta) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h} - \boldsymbol{v}^T \boldsymbol{b} - \boldsymbol{c}^T \boldsymbol{h}, \tag{3}$$

where $\boldsymbol{b} \in \mathbb{R}^M$ and $\boldsymbol{c} \in \mathbb{R}^N$ are the bias of the visible layer and hidden layer, respectively, and $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ is the mapping matrix. All model parameters together are denoted $\Theta = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$. We can easily use a tensor network diagram to represent Eq. 3 in Fig. 4 (a). The conditional distributions over the hidden and visible layers can be written as:

$$p(\boldsymbol{v} = \boldsymbol{1} | \boldsymbol{h}; \Theta) = \sigma(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{b}), \tag{4}$$

$$p(\boldsymbol{h} = \boldsymbol{1} | \boldsymbol{v}; \Theta) = \sigma(\boldsymbol{W}^T\boldsymbol{v} + \boldsymbol{c}), \tag{5}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic sigmoid function and $\boldsymbol{1}$ denotes a vector of ones. The parameter training in a standard RBM is commonly performed by the contrastive divergence (CD) algorithm [1] and its variants [12], [13].

D. Matrix-variate and tensor-variate RBMs

Here we briefly introduce the two non-vector RBM models, namely MvRBM [5] and TvRBM [6]. TvRBM employs a tensorial visible layer and keeps the hidden layer in a vector form. A rank-R CP tensor decomposition is used to connect the visible and hidden layers. However, such a connection form is also criticized to limit the model capability [5]. The corresponding energy function of TvRBM is shown in Fig. 4(b) as a tensor network diagram.

In the MvRBM model, both the input and hidden layers are matrices and are interconnected through two independent weight matrices $\boldsymbol{W}^{(1)} \in \mathbb{R}^{M_1 \times N_1}, \boldsymbol{W}^{(2)} \in \mathbb{R}^{M_2 \times N_2}$. This particular construction reduces the total number of parameters from $M_1 \times M_2 \times N_1 \times N_2$ down to $M_1 \times N_1 + M_2 \times N_2$ but comes at the cost of a limited representation ability, as the weight matrix $\boldsymbol{W}$ is constrained to be a Kronecker product of the $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ matrices. The energy function of the MvRBM is graphically represented as a tensor diagram in Fig. 4(c).

IV. MPORBM

We now describe the proposed MPORBM and its customized model parameter learning algorithms.

A. Model definition

In an MPORBM, both the visible layer $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ and the hidden layer $\boldsymbol{\mathcal{H}} \in \mathbb{R}^{J_1 \times \cdots \times J_d}$ are d-way tensors. As a result, the weight matrix $\boldsymbol{W}$ is now a $2d$-way tensor

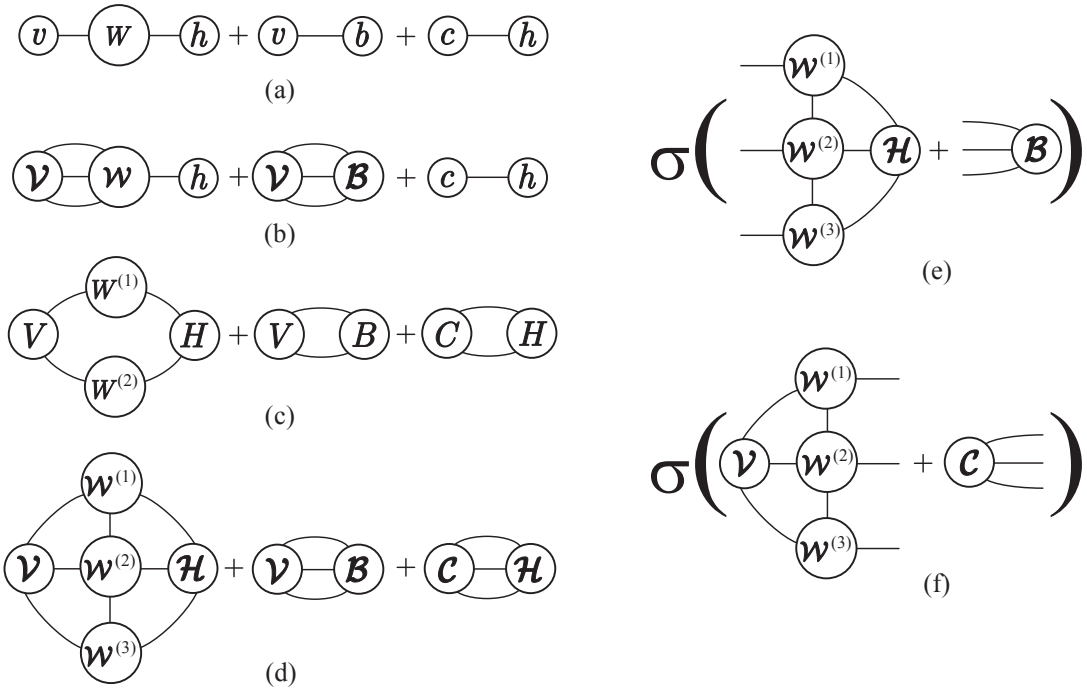

Fig. 4: Negative energy functions ($-E$) of (a) RBM; (b) TvRBM; (c) MvRBM; (d) MPORBM. And conditional probability of an MPORBM over its (e) hidden layer; (f) visible layer.

$\boldsymbol{\mathcal{W}} \in \mathbb{R}^{I_1 \times \cdots \times I_d \times J_1 \times \cdots \times J_d}$, which is represented by an MPO instead. With both the visible and hidden layers being tensors, it is therefore also required that the bias vectors $\boldsymbol{b}, \boldsymbol{c}$ are tensors $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}, \boldsymbol{\mathcal{C}} \in \mathbb{R}^{J_1 \times \cdots \times J_d}$, respectively. The energy function of an MPORBM is shown in Fig. 4(d) for the specific case where $d = 3$. The vertical edges between the different MPO-cores $\boldsymbol{\mathcal{W}}^{(1)}, \ldots, \boldsymbol{\mathcal{W}}^{(d)}$ are the key ingredient in being able to express generic weight tensors $\boldsymbol{\mathcal{W}}$, as opposed to the two disconnected weight matrices in the MvRBM model. The corresponding conditional distribution equations over the hidden and visible layers can be derived from the energy function and are visualized in Fig. 4(e)&(f) for the case where $d = 3$. This involves the summation of the weight MPO with either the hidden or visible layer tensors into a d-way tensor, which is then added element-wise with the corresponding bias tensor. The final step in the computation of the conditional probability is an element-wise application of the logistic sigmoid function on the resulting tensor.

B. MPORBM Learning algorithm

Let $\Theta = \{\boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{W}}^{(1)}, \boldsymbol{\mathcal{W}}^{(2)}, \ldots, \boldsymbol{\mathcal{W}}^{(d)}\}$ denote the model parameters. The model learning task is then formulated into maximizing the training data likelihood:

$$\boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{V}}; \Theta) = p(\boldsymbol{\mathcal{V}}; \Theta) = \sum_{\boldsymbol{\mathcal{H}}} p(\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{H}}; \Theta) \tag{6}$$

with respect to model parameter Θ. Similar to the standard RBM [1], the expression of the gradient of the log-likelihood is:

$$\frac{\partial}{\partial \Theta} \log \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{V}}; \Theta) = -\mathbb{E}_{\boldsymbol{\mathcal{H}}|\boldsymbol{\mathcal{V}}} \left[\frac{\partial E(\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{H}})}{\partial \Theta} \right] + \mathbb{E}_{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{H}}} \left[\frac{\partial E(\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{H}})}{\partial \Theta} \right] \tag{7}$$

where $\mathbb{E}_{\boldsymbol{\mathcal{V}}|\boldsymbol{\mathcal{H}}}$ is the data expectation w.r.t. $p(\boldsymbol{\mathcal{H}}|\boldsymbol{\mathcal{V}}; \Theta)$, which can be computed by Fig. 4(f). $\mathbb{E}_{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{H}}}$ is the model expectation w.r.t. $p(\boldsymbol{\mathcal{H}}, \boldsymbol{\mathcal{V}}; \Theta)$, which can be approximately computed by the CD procedure. The main idea in the CD procedure is as follows: first, a Gibbs chain is initialized with one particular training sample $\boldsymbol{\mathcal{V}}_{(0)} = \boldsymbol{\mathcal{X}}_{train}$. Figs. 4(e)&(f) are then computed K times in an alternating fashion, which results in the chain $\{(\boldsymbol{\mathcal{V}}_{(0)}, \boldsymbol{\mathcal{H}}_{(0)}), (\boldsymbol{\mathcal{V}}_{(1)}, \boldsymbol{\mathcal{H}}_{(1)}), \ldots, (\boldsymbol{\mathcal{V}}_{(K)}, \boldsymbol{\mathcal{H}}_{(K)})\}$. The model expectation is then approximated by $\{\boldsymbol{\mathcal{V}}_{(K)}\}$. The derivative of the log-likelihood with respect to the k-th MPO-core $\boldsymbol{\mathcal{W}}^{(k)}$ is visualized in Fig. 5. This involves the computation of 2 fourth-order tensors, obtained by removing the k-th MPO-core $\boldsymbol{\mathcal{W}}^{(k)}$ from the two tensor networks and summing over all connected edges. The resulting tensors are then subtracted element-wise from one another. The derivatives of the log-likelihood with respect to the bias tensors $\boldsymbol{\mathcal{B}}, \boldsymbol{\mathcal{C}}$ are

$$\frac{\partial}{\partial \boldsymbol{\mathcal{B}}} \log \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{V}}; \Theta) = \boldsymbol{\mathcal{V}}_{(0)} - \boldsymbol{\mathcal{V}}_{(K)},$$

$$\frac{\partial}{\partial \boldsymbol{\mathcal{C}}} \log \boldsymbol{\mathcal{L}}(\boldsymbol{\mathcal{V}}; \Theta) = \boldsymbol{\mathcal{H}}_{(0)} - \boldsymbol{\mathcal{H}}_{(K)}.$$

We mainly use the CD procedure to train the MPORBM model. However, instead of updating all the MPO-cores simultaneously with one batch of input training data, we employ

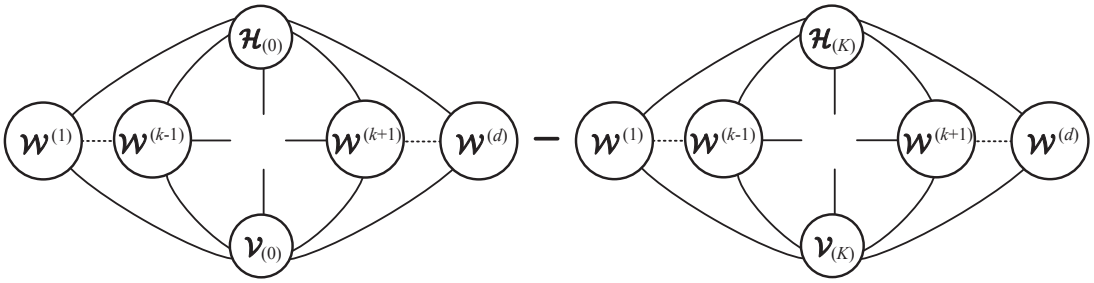

Fig. 5: The derivative of the log-likelihood function with respect to the kth MPO-core $\mathcal{W}^{(k)}$.

the alternating optimization procedure (AOP). This involves updating only one MPO-core at a time while keeping the others unchanged using the same batch of input training data. We name this parameter learning algorithm CD-AOP and its pseudo-code is given in Algorithm 1 . The superiority of this alternating optimization procedure over simultaneously updating all MPO-cores, which we call CD-SU from hereon will be demonstrated through numerical experiments.

Algorithm 1: MPORBM learning algorithm (CD-AOP)

Input: Training data of N tensors $\mathcal{D} = \{\mathcal{X}_1, \ldots, \mathcal{X}_N\}$, the maximum iteration number T, the batch size b, the momentum γ, the learning rate α and CD step K.

Output: Model parameters $\Theta = \{\mathcal{B}, \mathcal{C}, \mathcal{W}^{(1)}, \mathcal{W}^{(2)}, \ldots, \mathcal{W}^{(d)}\}$.

Randomly initialize $\mathcal{W}^{(1)}, \ldots, \mathcal{W}^{(d)}$. Set the bias $\mathcal{B} = \mathcal{C} = \mathcal{O}$ and the gradient increments $\Delta\mathcal{B} = \Delta\mathcal{C} = \Delta\mathcal{W}^{(1)} = \cdots = \Delta\mathcal{W}^{(d)} = \mathcal{O}$.
for iteration number $t = 1 \rightarrow T$ **do**
  Randomly divide $\mathcal{D}$ into M batches $\mathcal{D}_1, \ldots, \mathcal{D}_M$ of size b.
  for batch $m = 1 \rightarrow M$ **do**
    for $c = 1 \rightarrow d$ **do**
      For all data $\mathcal{V}_{(0)} = \mathcal{X}_{train} \in \mathcal{D}_m$ run Gibbs sampling:
      for k=0,...,K-1 **do**
        sample $\mathcal{H}_{(k)}$ according to Fig. 4 (f) with $\mathcal{V}_{(k)}$;
        sample $\mathcal{V}_{(k+1)}$ according to Fig. 4(e) with $\mathcal{H}_{(k)}$
      end for
      $\Delta\mathcal{W}^{(c)} \leftarrow \gamma\Delta\mathcal{W}^{(c)} + \alpha(\frac{1}{b}\sum_{\mathcal{D}_m}\frac{\partial}{\partial\mathcal{W}^{(c)}}\log\mathcal{L}(\mathcal{V};\Theta))$
      $\Delta\mathcal{B} \leftarrow \gamma\Delta\mathcal{B} + \alpha(\frac{1}{b}\sum_{\mathcal{D}_m}\frac{\partial}{\partial\mathcal{B}}\log\mathcal{L}(\mathcal{V};\Theta))$
      $\Delta\mathcal{C} \leftarrow \gamma\Delta\mathcal{C} + \alpha(\frac{1}{b}\sum_{\mathcal{D}_m}\frac{\partial}{\partial\mathcal{C}}\log\mathcal{L}(\mathcal{V};\Theta))$
      $\Theta \leftarrow \Theta + \Delta\Theta$
    end for
  end for
end for

V. Experiments

In this section, the MPORBM is compared with both the standard RBM and MvRBM. Specifically, we first investigate the performance and scalability of these models as a means to do feature extraction for classification, together with the influence of the MPO-ranks on the expressive power of the MPORBM. Furthermore, we demonstrate the generative capacity of MPORBM by implementing an image completion task and an image denoising task. For hyperparameter setting, we select mostly the same default values as in MvRBM paper, namely momentum γ = 0.5, CD step K = 1. For learning rate, we provide a set of possible value and choose the one which achieves the best validation accuracy. Moreover, we choose the same maximum iteration number and batch size for all methods. All experiments are run in MATLAB on an Intel i5 3.2GHz desktop with 16GB RAM.

TABLE I: Detailed information for different datasets.

Datasets	Original data size	Data value range
Alphadigits	20 x 16	Binary
DrivFace	80 x 80	0-255 integer
Arcene	10000 x 1	0-924 integer
COIL-100	128 x 128 x 3	0-255 integer

TABLE II: Visual layer structure of different RBM models for different datasets.

Datasets	RBM	MvRBM	MPORBM
Alphadigits	320 x 1	20 x 16	20 x 16
DrivFace	51200 x 1	80 x 640	80 x 80 x 8
Arcene	100000 x 1	100 x 1000	100 x 100 x 10
COIL-100	393216 x 1	128 x 3072	128 x 128 x 24

A. Data Classification

In the first experiment, we demonstrate the data classification accuracy superiority of MPORBM on extensive standard machine learning datasets, namely the Binary Alphadigits dataset*, normalized DrivFace†, Arcene‡ and COIL-100 dataset§. The size of those datasets vary from 320 to 49152. Since we assume binary input in our RBM setting, thus for non-binary datasets, we first use a multi-bit vector to represent each value in the original data. The additional multi-bit dimension is combined with the RGB channel if the dataset is color image. Table I shows the detailed information of these datasets, while Tables II and III show the visual and hidden layer structure of different RBM models for different datasets.

*https://cs.nyu.edu/~roweis/data.html
†https://archive.ics.uci.edu/ml/datasets/DrivFace
‡https://archive.ics.uci.edu/ml/datasets/Arcene
§http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php

TABLE III: Hidden layer structure of different RBM models for different datasets.

Datasets	RBM	MvRBM	MPORBM
Alphadigits	80 x 1	10 x 8	10 x 8
DrivFace	500 x 1	10 x 50	10 x 10 x 5
Arcene	500 x 1	10 x 50	10 x 10 x 5
COIL-100	500 x 1	10 x 50	10 x 10 x 5

TABLE IV: Classification errors of different RBM models.

Datasets	RBM	MvRBM	MPORBM CD-SU/AOP
Alphadigits	28.10%	31.20%	28.10% / 26.90%
DrivFace	24.20%	15.48%	9.68% / 8.06%
Arcene	45.00%	34.00%	32.00% / 27.00%
COIL-100	–	6.82%	6.82% / 0.00%

We randomly separate the above four datasets into training, validation and testing parts respectively. It might happen that training sample sizes are insufficiently large to train a good RBM model. Thus the training data number we choose for each class is significantly smaller than their data dimension, commonly smaller than 35. The MPORBM-ranks were chosen empirically from a range of 10 to 50 depending on the model input size. The trained RBM models were then employed to extract features from the hidden layer and then those new features were utilized to train a K Nearest Neighbor (K-NN) classifier with $K = 1$ for all experiments. Table IV lists the resulting classification errors.

The restrictive Kronecker product assumption of the weight matrices in MvRBM explains why it has the worst classification performance in all datasets. The standard RBM also performs worse than MPORBM, which may be caused by overfitting because of the small training sample size. It is notable that due to the PC memory constraint, the standard RBM fails to learn the enormous number of parameters in the full weight matrix for COIL-100 dataset, but MPORBM still gets the best classification performance due to the general MPO weight structure. Moreover, the CD-AOP algorithm shows a higher classification accuracy than CD-SU, which further indicates that the alternating updating scheme is more suitable for our MPORBM model. Thus in the following experiments, we only employ CD-AOP algorithm to train our MPORBM model.

B. Investigation of influence MPO-rank on classification error

In this experiment, we demonstrate how the expressive power of the MPORBM is determined by the MPO-ranks. For this purpose, the MNIST dataset[¶] was used to train an MPORBM for image classification. The MNIST dataset has a training set of 60k samples and a test set of 10k samples. Each sample is a 28×28 grayscale picture of handwritten digits $\{0, \ldots, 9\}$, where each pixel value was converted into a binary number. The hidden layer states in the RBM model were also regarded as features extracted from the training data and used in a K-NN classifier with $K = 1$. The dimensions of the hidden layer were set to $M_1 = M_2 = 10$. The MPO-rank R_2 was varied from 2 up to 200. To reveal the general rule of MPORBM model expression power on MPO-ranks, the above mentioned experiments were run for training sample sizes of 3k, 4k and 5k, which are randomly chosen from training set. In addition, 10k samples of the training set were chosen for validation. The classification error for each of these runs is shown in Fig. 6 as a function of R_2. Note that the MPORBM with $R_2 = 1$ is identical with the MvRBM for these matrix inputs. It can be seen that the classification error stabilizes when $R_2 \approx 40$, which indicates that a low-rank MPO may be powerful enough to model the latent probability distribution of the training data and a lot of storage and computational resources can be saved. We need to mention that by setting the MPO-ranks to their maximal values the MPORBM gets the same model expression ability as a standard RBM. This experiment, however, shows that using a low-rank MPORBM is more than enough to model real-life data.

[¶]http://yann.lecun.com/exdb/mnist/

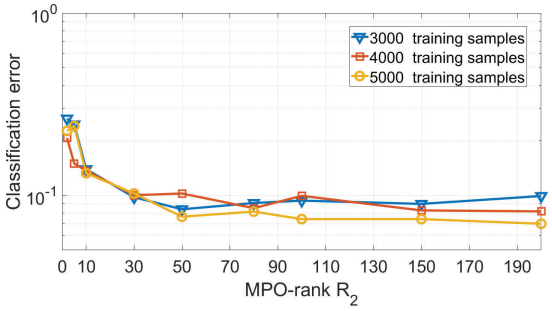

Fig. 6: Classification error as a function of the MPO-rank R_2 in the MPORBM model.

C. Image completion and denoising

In this experiment, we show that an MPORBM is good at generative modeling by implementing image completion and denoising. We test the above generative tasks on the binarized MNIST dataset. Specifically, we randomly choose 50 samples from the 60k training set to train the RBM models with 500 iterations and using all 10k test samples to check the model generative performance. The standard RBM is set up with vectorial visual layer and hidden layer, while both MvRBM and MPORBM are constructed with 2-way tensorial visual layer and hidden layer. The MPO-rank in this experiment is set to 40 according to the observation from the previous experiment. The visual-hidden layer structure and total weight parameter number of each RBM model is listed in Table V.

In the image completion task, one half of the image is provided to the trained RBM models to complete the another

TABLE V: The visual-hidden layer structure and total weight parameter number of different RBM models.

	RBM	MvRBM	MPORBM CD-AOP
Visual structure	784 x 1	28 x 28	28 x 28
Hidden structure	100 x 1	10 x 10	10 x 10
# parameters	78400	560	22400

Fig. 7: Image completion result when given the right half image. First row: original binarized images; second row: RBM completed images; third row: MvRBM completed images; fourth row: MPORBM completed images.

Fig. 8: Image completion result when given the bottom half image. First row: original binarized images; second row: RBM completed images; third row: MvRBM completed images; fourth row: MPORBM completed images.

half. As mentioned in [14], RBM completion performance is different when the given half image is in the column or row direction. Thus we investigate the completion ability of different RBM models when given the right and the bottom half of image, respectively. Figures 7 and 8 demonstrate the completed images of different RBM models when given the same randomly selected right and bottom halves, respectively. It is clear that MvRBM is not able to complete the image, which further confirms the necessity of the MPO generalization. Table VI further lists the average PSNR results between the completed image and the original binarized image on the whole test set. We found that the MPORBM demonstrates a comparable image completion performance to a standard RBM, but with much fewer model parameters (around 29%).

In the image denoising task, we employ the same trained RBM models from the completion task and randomly add a certain percentage of salt & pepper noise to the test set. The average PSNR results between the original binarized pictures and denoised pictures on the whole test set are listed in Table VII. It is clear to see that the denoising performance of the MPORBM is comparable with that of a standard RBM when 10% noise is added, but performs better when higher percentages of noise are added. The fewer model parameters in the MPORBM may lead to a more robust generative model.

TABLE VI: The average PSNR results on the whole test set when given the right and bottom half images respectively, and completing the left and upper half images.

Given half	RBM	MvRBM	MPORBM CD-AOP
Right half	13.86 dB	12.35 dB	13.79 dB
Bottom half	13.66 dB	12.38 dB	13.69 dB

TABLE VII: The average PSNR results on the whole test set when adding $p\%$ salt & pepper noises.

$p\%$ noise	RBM	MvRBM	MPORBM CD-AOP
10%	13.55 dB	11.82 dB	13.49 dB
15%	12.90 dB	10.26 dB	13.24 dB
20%	11.88 dB	9.23 dB	12.95 dB

VI. Conclusion

This paper has proposed the MPORBM, which preserves the tensorial nature of the input data and utilizes a matrix product operator (MPO) to represent the weight matrix. The MPORBM generalizes all existing RBM models to tensor inputs and has better storage complexity since the number of parameters grows only linearly with the order of the tensor. Furthermore, by representing both the visible and hidden layers as tensors, it is possible to retain useful structural information in the original data. In order to facilitate the exposition on the various computations, tensor network diagrams were introduced and used throughout the paper.

There are several avenues for future research. If both the visible and hidden layers are represented in a matrix product state (MPS) form, then all computations can be done on individual cores. This can significantly improve the computational complexity of the learning algorithm.

Furthermore, analogous to stacking RBMs into a Deep Belief Network (DBN) or Deep Belief Machine (DBM), MPORBMs can be stacked into an MPODBN or MPODBM to extend its expressive power and application. Again, this stacking can also gain computational benefits from keeping all layers in the MPS form.

References

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

[2] H. Larochelle and Y. Bengio, "Classification using discriminative restricted Boltzmann machines," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 536–543.

[3] A. Krizhevsky, G. Hinton *et al.*, "Factored 3-way restricted Boltzmann machines for modeling natural images," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 621–628.

[4] A.-r. Mohamed and G. Hinton, "Phone recognition using restricted Boltzmann machines," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 4354–4357.

[5] G. Qi, Y. Sun, J. Gao, Y. Hu, and J. Li, "Matrix variate restricted Boltzmann machine," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 389–395.

[6] T. D. Nguyen, T. Tran, D. Q. Phung, S. Venkatesh *et al.*, "Tensor-Variate Restricted Boltzmann Machines." in *AAAI*, 2015, pp. 2887–2893.

[7] R. Orús, "A practical introduction to tensor networks: Matrix product states and projected entangled pair states," *Annals of Physics*, vol. 349, pp. 117–158, 2014.

[8] G. W. Taylor and G. E. Hinton, "Factored conditional restricted Boltzmann machines for modeling motion style," in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 1025–1032.

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

[10] J. Chen, S. Cheng, H. Xie, L. Wang, and T. Xiang, "Equivalence of restricted Boltzmann machines and tensor network states," *Phys. Rev. B*, vol. 97, p. 085104, Feb 2018.
[11] I. V. Oseledets, "Tensor-train decomposition," *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2295–2317, 2011.
[12] T. Tieleman, "Training restricted Boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1064–1071.
[13] T. Tieleman and G. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1033–1040.
[14] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, "Unsupervised generative modeling using matrix product states," *Physical Review X*, vol. 8, no. 3, p. 031012, 2018.