

**Tree tensor networks for generative modeling**Song Cheng,<sup>1,2,\*</sup> Lei Wang,<sup>1,†</sup> Tao Xiang,<sup>1,‡</sup> and Pan Zhang<sup>3,§</sup><sup>1</sup>*Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China*<sup>2</sup>*University of Chinese Academy of Sciences, Beijing, 100049, China*<sup>3</sup>*CAS Key Laboratory of Theoretical Physics, Institute of Theoretical Physics, Chinese Academy of Sciences, Beijing 100190, China*

(Received 8 January 2019; revised manuscript received 4 April 2019; published 18 April 2019)

Matrix product states (MPSs), a tensor network designed for one-dimensional quantum systems, were recently proposed for generative modeling of natural data (such as images) in terms of the “Born machine.” However, the exponential decay of correlation in MPSs restricts its representation power heavily for modeling complex data such as natural images. In this work, we push forward the effort of applying tensor networks to machine learning by employing the tree tensor network (TTN), which exhibits balanced performance in expressibility and efficient training and sampling. We design the tree tensor network to utilize the two-dimensional prior of the natural images and develop sweeping learning and sampling algorithms which can be efficiently implemented utilizing graphical processing units. We apply our model to random binary patterns and the binary MNIST data sets of handwritten digits. We show that the TTN is superior to MPSs for generative modeling in keeping the correlation of pixels in natural images, as well as giving better log-likelihood scores in standard data sets of handwritten digits. We also compare its performance with state-of-the-art generative models such as variational autoencoders, restricted Boltzmann machines, and PixelCNN. Finally, we discuss the future development of tensor network states in machine learning problems.

DOI: [10.1103/PhysRevB.99.155131](https://doi.org/10.1103/PhysRevB.99.155131)**I. INTRODUCTION**

Generative modeling [1], which is being asked to learn a joint probability distribution from training data and generate new samples according to it, is a central problem in unsupervised learning. Compared with discriminative modeling, which captures only the conditional probability of the data’s discriminative labels, generative modeling attempts to capture the whole joint probability of the data and is therefore much more difficult [2].

During the past decades, there have been many generative models proposed, including those based on the probabilistic graphic model (PGM), such as the Bayesian network [3], the hidden Markov model [4], and restricted Boltzmann machines (RBM) [5] and models based on neural networks such as deep belief networks [6], variational autoencoders (VAEs) [7], RealNVP [8], PixelCNN [9,10], and the recently very popular generative adversarial networks [11]. Among these generative models, two models are motivated by physics. One is the Boltzmann machine [12], where the joint distribution is represented by Boltzmann distribution; the other one is the Born machine, where Born’s rule in quantum physics is borrowed to represent the joint probability distribution of data with the square amplitude of a wave function [13–16] and the wave function is represented by tensor networks.

Tensor networks (TNs) were originally designed for efficiently representing quantum many-body wave functions [17,18], which, in general, are described by a high-order tensor with exponential parameters. A TN applies low-rank decompositions to the general tensor by discarding the vast majority of unrelated long-range information to break the so-called exponential wall of quantum many-body computation. Popular TNs include matrix product states (MPSs) [19], tree tensor networks (TTNs) [20], the multiscale entanglement renormalization ansatz [21], projected entanglement pair states (PEPSs) [22], etc.

In recent years, researchers have begun to notice the similarities between the tensor networks and the PGM [23,24]. Specifically, the factor graph in the PGM can be seen as a special kind of tensor network [25]. In addition to the structural similarities, the problems faced by the TN and PGM are also similar. They both try to use few parameters to approximate the probability distribution of an exponentially large number of parameters. The reason TNs can achieve this is attributed to the physical system’s locality. That is, most of the entanglement entropy of the quantum states we care about obeys the area law [26]. On the PGM side, although the success of machine learning models based on PGM for natural images is not completely understood, some arguments support the idea that natural images actually have only sparse long-range correlations, making them much less complex than arbitrary images [13,27]. Thus, physicist-favored quantum states and natural images may both gather only in a tiny corner of their possible space, and the similarity between TN and ML models may essentially result from the similarity of the intrinsic structures of the model and data. Building upon this similarity, various works have emerged in recent

\*physichengsong@iphy.ac.cn

†wanglei@iphy.ac.cn

‡txiang@iphy.ac.cn

§panzhang@itp.ac.cn

years that apply the concept [23,24,28,29], structure [30–32], and algorithm [14,33,34] of the tensor networks to machine learning.

In this work, we focus on generative modeling based on tensor networks. On the one hand, we propose TTNs as a direct extension to the tree-structure factor graph models. On the other hand, a TTN works as a new tensor network generative model, an extension of the recently proposed MPS Born machine [14]. Compared to MPSs, TTNs exhibit natural modeling on two-dimensional data such as natural images, and its more favorable in the growth of the correlation length of pixels.

In this paper we first introduce the TTN as a generative model, then develop an efficient sweeping algorithm to learn the model from data by minimizing the Kullback-Leibler divergence between empirical data distribution and the model distribution, as well as a sampling algorithm that generates unbiased samples. We apply our TTN generative model to two kinds of data. The first one is random binary patterns, where the TTN model works as an associative memory trying to remember all the given patterns. The task is to test the expressive power of the TTN model. The second type of data we test is the MNIST data set of handwritten digits, a standard data set in machine learning. Using extensive numerical experiments, we show that the TTN has better performance than the classic tree-structure factor graph and the MPS Born machine. In addition, we demonstrate quantitatively the gap between the existing tensor network generation models and state-of-the-art machine learning generative models, pointing out the possible future development of the tensor network generation model.

The rest of the paper is organized as follows: in Sec. II we give a detailed description of the TTN model, a two-dimensional structure construction, and the training and generating algorithms. In Sec. III we apply the TTN model to both the binary random patterns and the standard binary MNIST data set. Finally, we discuss the future of tensor networks applied to unsupervised generative learning in Sec. IV.

## II. MODELS AND ALGORITHMS

### A. Data distribution and maximum-likelihood learning

Suppose we are given a set of data composed of  $|\mathcal{T}|$  binary images,  $\{\mathbf{x}_a | a = 1, 2, 3, \dots, |\mathcal{T}|\} \in \{+1, -1\}^{|\mathcal{T}| \times n}$ , each of which is represented by a binary vector of length  $n$ . This defines an empirical data distribution:

$$\pi(\mathbf{x}) = \frac{1}{|\mathcal{T}|} \sum_{a=1}^{|\mathcal{T}|} \delta(\mathbf{x}, \mathbf{x}_a).$$

The task of generative modeling is to find an efficient way to model  $\pi(\mathbf{x})$ , which means finding a distribution  $p(\mathbf{x})$  (with a reasonable number of parameters) which is as close as possible to  $\pi(\mathbf{x})$ . The distance between those two probabilities can be defined by using the Kullback-Leibler (KL) divergence [35],

$$D_{\text{KL}}(\pi \| p) = \sum_{\mathbf{x}} \pi(\mathbf{x}) \ln \left( \frac{\pi(\mathbf{x})}{p(\mathbf{x})} \right).$$

We hence introduce the negative log likelihood (NLL) as the cost function for model learning:

$$\mathcal{L} = -\frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \ln[p(\mathbf{x})] = S(\pi) + D_{\text{KL}}(\pi \| p), \quad (1)$$

where  $\mathcal{T}$  indicates the set of given data and  $|\mathcal{T}|$  is the number of training images. Due to the non-negativity of the KL divergence, the last equation indicates that the NLL is bounded below by the Shannon entropy of the data set. Moreover, since the Shannon entropy  $-\sum \pi(\mathbf{x}) \ln \pi(\mathbf{x})$  is independent of models, minimizing the NLL is equivalent to minimizing the KL divergence.

### B. Tree-structure factor graph as a generative model

The art of generative modeling is deeply related to determining a good architecture when representing the best joint probability  $p(\mathbf{x})$ , which enhances the generalizability. Considering the difficulty of calculating the normalization factor of a loop graph, a loop-free PGM like a chain or a tree is always a relatively simple starting point. Here we take the tree-structure factor graph as an example. The unnormalized joint probability  $\tilde{p}(\mathbf{x})$  in a tree-structure factor graph is represented as

$$\begin{aligned} P(\mathbf{x}) &= \frac{1}{Z} \tilde{p}(\mathbf{x}) \\ &= \frac{1}{Z} \sum_{\{h_1, h_2, \dots, h_{N-1}\}} f^1(h_1, h_2) f^2(h_2, h_3) \cdots f^{2N-2} \\ &\quad \times (h_{N-1}, x_N). \end{aligned}$$

As shown in Fig. 1(a), each block represents a random variable with two states,  $\{+1, -1\}$ . Each purple node  $i$  is called a visible node, whose state  $x_i$  is determined by the value of one pixel of the binary input data. Blue node  $h_j$  also has two states, but they act as hidden variables and hence are not supposed to be observed directly. Each edge of the graph introduces an arbitrary function  $f^k$  which maps the states of two-end-point nodes into a scalar. By combining the scalar on all factors and summing over all possible states of hidden variables  $\mathbf{h}$ , one gets the non-normalized probability  $\tilde{p}(\mathbf{x})$  for a given configuration of pixels  $\mathbf{x}$ .

The learning is processed by using gradient descent to minimize the NLL for the given data set. By denoting the learnable parameters of the model as  $\theta$ , the gradients read

$$-\nabla_{\theta} \mathcal{L} = -\frac{1}{|\mathcal{T}|} \sum_{\mathbf{x} \in \mathcal{T}} \nabla_{\theta} \ln \tilde{p}_{\theta}(\mathbf{x}) + \nabla_{\theta} \ln Z. \quad (2)$$

In general the first term in the above equation is relatively straightforward to compute. However, computing the second term requires summing over  $2^n$  configurations and hence is difficult to compute in general. Fortunately, for acyclic graphs such as the tree-structure factor graph, the second term can be computed exactly by using the sum-product algorithm [36]; a generic message passing algorithm operates on the factor graphs. It is a simple computational rule that by exchanging the multiplication and summation in  $Z$  with a certain order we can avoid the exponential problem in the brute-force summation.

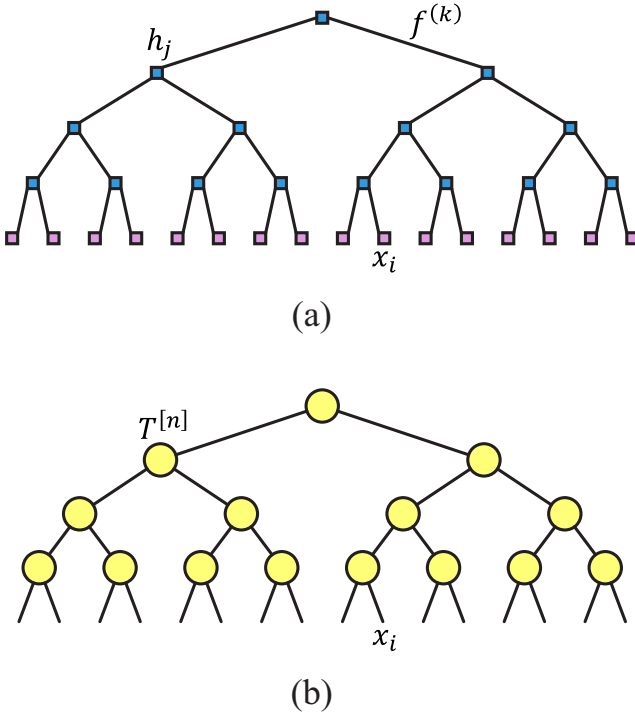


FIG. 1. (a) Tree-structure factor graph, where each block denotes a random variable with a value of  $-1$  or  $1$ , in which the blue (purple) block represents a hidden (visible) variable. The edge between two blocks introduces a factor function  $f^{(k)}$  of those two variables. By adjusting those factor functions, the model could obtain the appropriate joint probability. (b) Tree tensor network, where  $x_i$  denote the value of the  $i$ th pixel of the data set. Each yellow circle denotes a two- or three-order tensor. The edge between two tensors denotes a share index of tensors, which is also called a virtue index in the literature and will be contracted later. The exposed edge denotes the so-called physical index of tensors; those indices would ultimately be determined by the data set. For one of the given configurations of the physical indices, the probability of the configuration is proportional to the final scale value of the TTN after contracting all the virtue indices.

It has been proved that any factor graph can be mapped to a tensor network, whereas only a special type of tensor network has corresponding factor graphs [25]. We take the tree-structure graph model as an example. Let us put a matrix  $M^{(k)}$  in each edge  $k$  and an identity tensor  $\delta^{(j)}$  in each hidden node  $h_j$ , with the elements being written as

$$M_{h_a, h_b}^{(k)} = f^k(h_a, h_b) \quad (3)$$

and

$$\delta_{l,r,u}^{(j)} = \begin{cases} 1, & l = r = u, \\ 0, & \text{otherwise,} \end{cases} \quad (4)$$

where each index of  $\delta^{(j)}$  corresponds to an adjacent edge of  $h_j$  and bond dimensions of those indices are identical to the number of states of  $h_j$ . One can use either QR decomposition or singular-value decomposition (SVD) decomposition to separate  $M^{(k)}$  into a product of two matrices as

$$M_{h_a, h_b}^{(k)} = \sum_k A_{h_a, k}^{(k)} B_{k, h_b}^{(k)}. \quad (5)$$

Without loss of generality, here we assume in the graph  $h_a \geq h_b$ . The obtained matrices  $A$  and  $B$  can be absorbed into a tensor defined on nodes,

$$T_{l,r,u}^{(j)} = \sum_{x,y,z} B_{l,x}^{(l)} B_{r,y}^{(r)} \delta_{x,y,z}^{(j)} A_{z,u}^{(u)}. \quad (6)$$

For  $j = 1$ , we simply let the bond dimension of  $z, u$  equal 1. Now we arrive at a specific form of TNN as shown in Fig. 1(b). Notice that the tensor  $T^{(j)}$  here is just a special subset of the general three-order tensor, which means if we use general tensors as the building blocks of the TNN, we would get an extension of the origin factor graph model.

Here we want to recall that the rule of the sum-product approach in a tree-structure factor graph is, in fact, equivalent to the tensor contraction of the TTN, with the same order that the sum-product algorithm applies. However, notice that the tensor contraction is much more general than the sum-product algorithm. In the cases when the sum-product algorithm is no longer applicable, the TN can still be approximately contracted using approaches such as the tensor renormalization group [37].

### C. Tree tensor network generative model

As motivated in the last section, we treat the TTN as a direct extension of the tree-structure factor graph for generative modeling. As illustrated in Fig. 1(b), each circle represents a tensor; each edge of the circle represents an individual index of the tensor. The first tensor is a matrix connecting the second and third tensors, while the remaining tensors are all three-order tensors with three indices. The index between two tensors is called a virtual bond, which would be contracted hereafter. The left and right indices of the tensors in the bottom of the TTN are respectively connected to two pixels of the input image and hence are called physical bonds.

As we indicated in the Introduction, the TTN generative model can also be treated as one kind of Born machine [13]; that is, the TTN represents a pure quantum state  $\Psi(\mathbf{x})$ , and  $p(\mathbf{x})$  is induced by the square of the amplitude of the wave function following Born's rule,

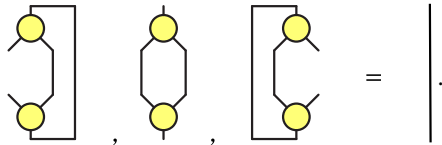
$$p(\mathbf{x}) = \frac{|\Psi(\mathbf{x})|^2}{Z}, \quad (7)$$

where  $Z = \sum_{\mathbf{x}} |\Psi(\mathbf{x})|^2$  is the normalization factor. In the TTN,  $\Psi(\mathbf{x})$  is represented as a contraction of the total  $N_t$  tensors in the TTN,

$$\Psi(\mathbf{x}) = \sum_{\{\alpha\}} T_{\alpha_2, \alpha_3}^{[1]} \prod_{n=2}^{N_t} T_{\alpha_n, \alpha_{2n}, \alpha_{2n+1}}^{[n]}. \quad (8)$$

The reason we choose the quantum-inspired Born machine instead of directly modeling a joint probability is based on a belief that the Born machine representation is more expressive than classical probability functions [13,31]. Meanwhile, treating the TN as a quantum state could introduce the canonical form of the TN, which simplifies the TN contraction calculation and makes contractions more precise. For example, if tensor  $T^{[2]}$  fulfills  $\sum_{\alpha_4, \alpha_5} T_{\alpha_2, \alpha_4, \alpha_5}^{[2]} T_{\alpha_2', \alpha_4, \alpha_5}^{[2]} = \delta_{\alpha_2, \alpha_2'}$ , we say that the tensor  $T^{[2]}$  is canonical for index  $\alpha_2$ , or, more visually speaking, upper canonical. In the TTN, there are three kinds

of canonical forms for each tensor: upper canonical, left canonical, and right canonical, depending on which index was finally left. The three canonical forms are shown in the following diagrammatic notation:



The line on the right side represents the identity matrix.

It is technically easy to canonicalize a tensor in the TTN. For example, we can start from one end of the tree and use the QR decomposition of the tensor to push the noncanonical part of the tensor to the adjacent tensor. By repeating this step, finally, one will push all noncanonical parts of the TTN to just one tensor, called the central tensor, and all other tensors are in one of the three canonical forms. Analogous to the mixed-canonical form of MPSs, we call this form the mixed-canonical form of the TTN.

Once the TTN is in the canonical form, many calculations become simple, for example, the normalization factor  $Z$  finally becomes the squared norm of a tensor:

(9)

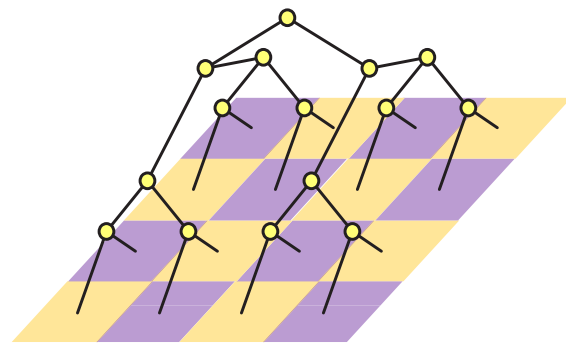
where the orange tensor represents the noncanonical central tensor in an arbitrary position. The direction of all the tensors' canonical forms is pointed toward the direction of the central tensor. After all, to get the normalization  $Z$ , the only calculation we need to do is the trace of multiplication of the central tensor by its complex conjugate.

General tensor networks have a gauge degree of freedom on their virtual bond. One can insert a pair of unitary matrices  $UU^{-1}$  in the virtual bond without changing the final contraction results. This could damage the accuracy of the training algorithm and brings additional computational complexity. Fortunately, for acyclic tensor networks like the TTN, the canonical form fixes this degree of freedom.

#### D. Data representations

In this work, we consider binary data, such as black and white images, so the local dimension of the Hilbert space of each physical bond is 2. As illustrated in Fig. 2, each index for the lowest-layer tensors has two components, corresponding to the two possible values of the connected pixels. The pixels can be simply vectorized from the image to a vector, as explored in [14] for the MPS Born machine, which we call *one-dimensional (1D) representation*, as it basically does not use any features in the two-dimensional (2D) structure of the images.

Compared with the MPS, a significant advantage of the TTN is that it can easily achieve the two-dimensional



(a)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

(b)

1	3	9	11
2	4	10	12
5	7	13	15
6	8	14	16

(c)

FIG. 2. (a) The TTN with 2D structure. Changing the 1D order of data with the 2D order is equivalent to using the TTN with 2D structure replacing Fig. 1(b). (b) The 1D order of data. (c) The 2D order of data.

modeling of natural images. Figure 2(a) shows the two-dimensional modeling of the TTN. In this architecture, each tensor is responsible for one local area of pixels, which greatly reduces the artificial fake long-range correlations. Hence, we call it the *2D representation*. Clearly, the 2D representation keeps the model structure of Fig. 1, while only requiring reshuffling the data index to proper order, as shown in Figs. 2(b) and 2(c) [21,38].

In practice, in order to ensure that the number of input pixels is a power of 2, we may artificially add some pixels that are always zero. If the input data are the 1D permutation, we add those zero pixels to the two ends of the one-dimensional chain; if it is 2D, we add to the outermost edge of the 2D lattice. This is analogous to the “padding” operation in convolution networks.

#### E. Training algorithm of the TTN

As we introduced in Sec. II A, the cost function we used in the training is the negative log likelihood [Eq. (1)], which is also the KL divergence between the target empirical data distribution and the probability distribution of our model, up to a constant.

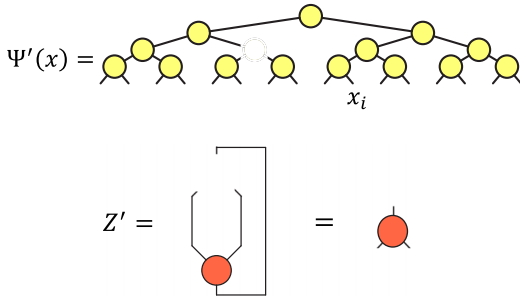
A standard way to minimize the cost function is the stochastic gradient descent algorithm (SGD). Unlike the traditional SGD, which updates all trainable parameters at the same time, in the TTN we have a sweeping process; that is, it iteratively updates each tensor based on the gradient



of the cost function with respect to the tensor elements of a tensor while holding other tensors unchanged. This sweeping process can be combined with the canonicalization form of the tensor network to simplify computations. As formulated in Eqs. (9) and (11), after canonicalization, the whole network is equivalent to one single tensor, which significantly improves the efficiency of the sweeping process. There are two choices for the updating scheme: a single-site update scheme in which we update a single three-way tensor at one time, with other tensors being held, and a two-site update scheme in which we first merge two neighboring tensors, then update the merged four-way tensors, with other tensors being held. For the single-site update, the gradient reads

$$\frac{\partial \mathcal{L}}{\partial T^{[k]}} = \frac{Z'}{Z} - \frac{2}{|T|} \sum_{\mathbf{x} \in T} \frac{\Psi'(\mathbf{x})}{\Psi(\mathbf{x})}, \quad (10)$$

where  $\Psi'(\mathbf{x})$  and  $Z'$  denote the derivatives of  $\Psi(\mathbf{x})$  and  $Z$  with respect to  $T^{[k]}$ ; they are depicted as



$$\Psi'(x) = \dots \quad (11)$$

$$Z' = \dots = \dots$$

As already noted, thanks to the tree canonicalization, computation of  $Z$  becomes straightforward.

The first step of the training is to transform the TTN consisting of many random initialized tensors into the canonical form and then push the noncanonical part onto the first tensor to be trained, e.g., the rightmost one. Next, we use the gradient calculated by Eqs. (10) and (11) to update corresponding tensors:

$$T_{\text{new}}^{[k]} = T^{[k]} - \alpha \frac{\partial \mathcal{L}}{\partial T^{[k]}}, \quad (12)$$

where  $\alpha$  denotes the learning rate; then we move to the next sweeping step. To maintain the canonical form, as shown in Algorithm 1, each time we apply a QR decomposition to the updated tensor, we store the orthogonal matrix  $Q$  as a new tensor and contract  $R$  to the tensor which is going to be updated in the next step.

If we start from the rightmost tensor of the TTN, this rule will allow us to gradually traverse the entire TTN from right to left. Then we choose the leftmost tensor as our starting tensor, doing the entire update again from left to right. A complete left-right-left sweeping defines an epoch of learning. See Algorithm 1 for the details of the training

algorithm.

---

### Algorithm 1 Sweeping algorithm of the TTN.

---

**Input:** Tensors  $T^{[l]}$  in the TTN. The TTN has been canonicalized towards the rightmost tensor  $T^{[N]}$ .  
**Output:** Updated tensor  $T_{\text{new}}^{[l]}$ . The TTN will be canonicalized towards the rightmost tensor  $T_{\text{new}}^{[N]}$ .  
 1: Mark all tensors as “unupdated.” Set  $T^{[N]}$  as the current tensor  $T^c$ .  
 2: **while** Exist unupdated tensors **do**  
 3:     **if** Exists one unupdated adjacent tensor of  $T^c$ , **then**  
 4:         Update  $T^c$  by the SGD. Mark this tensor as “updated.”  
 5:         Set the rightmost unupdated adjacent tensors of  $T^c$  as the next  $T^c$ .  
 6:         Apply QR decomposition on the previous  $T^c$ . Reshape  $Q$  to the shape of the previous  $T^c$ ; save it as  $T_{\text{new}}$ . Contract  $R$  to next  $T^c$ .  
 7:     **else if** Exist two unupdated adjacent tensors of  $T^c$ , **then**  
 8:         Do 5–6.  
 9:     **end if**  
 10: **end while**  
 11: Mark all tensors as “unupdated.”  
 12: Sweep from left to right.

---

For the two-site update, most of the procedures are the same as those for the single-site update. The only difference is that the tensor to be updated is a four-way tensor  $M^{[k,j]}$  merged by two three-way tensors. After using the gradient of  $\mathcal{L}$  on the merge tensor to update the merge tensor, we apply SVD on the merge tensor to rebuild two tensors with sizes identical to the original two tensors before merging while pushing the noncanonical part onto the next tensor. Each SVD step gives us a chance to change the dimension of the bond between the current tensor and the last tensor, making the TTN support dynamical adjustment of the number of parameters of the model. This is the main benefit of the two-site update scheme compared to the one-site update one. It is also an important advantage of the tensor network compared to traditional machine learning algorithms.

Notice that the one-site update always has lower computational complexity [ $O(D^3)$ ] than the two-site update [ $O(D^5)$ ]. In our experience, although the one-site update needs more epochs to converge, its final convergence result is not significantly different from the two-site update.

### F. Direct sampling of the TTN generative modeling

Unlike most of the traditional generative models, the TTN can directly calculate the partition function exactly. This gives the TTN the ability to sample configurations directly without needing the Markov chain, i.e., Gibbs sampling. We first compute the marginal probability of an arbitrary pixel  $k$ :

$$p(x_k) = \frac{\sum_{\mathbf{x}_a, \forall i \neq k} |\Psi(\mathbf{x})|^2}{Z}, \quad (13)$$

where the numerator in graphical notation is quite similar to that of computing  $Z$ , with the only difference being that the bond corresponding to  $x_k$  does not contract and  $\Psi(\mathbf{x})$  is

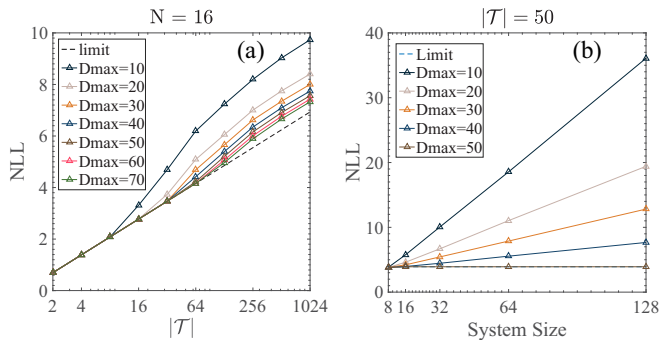


FIG. 3. (a) Training the NLL of the TTN Born machine as a function of the data size  $|\mathcal{T}|$ ; the system size is  $N = 16$ . (b) Training the NLL of the TTN Born machine as a function of the system size  $N$ ; the data size  $|\mathcal{T}| = 50$ .

left as a two-dimensional vector. The square of the values of this two-dimensional vector is the marginal probability of  $x_k = 0, 1$ . Then the conditional probability for the next pixel is computed as

$$p(x_j|x_k) = \frac{p(x_j, x_k)}{p(x_k)}. \quad (14)$$

In diagram notation this is equivalent to using a sampled value of  $x_k$  to fix the corresponding bond of  $x_k$  and keeping the corresponding bond of  $x_j$  open in contraction. The conditional probability of Eq. (14) can be generalized to the case of multiple fixed pixels. Equipped with all the conditional probabilities, we are able to sample pixels of images one by one.

### III. NUMERICAL EXPERIMENTS

#### A. Random data set

Remembering a specific set of random samples, i.e., as an associative memory [39], is perhaps the hardest and least biased task for testing the expressive power of generative models. Since in the TTN we are able to compute the partition function, the normalized probability of the training sample, and the NLL exactly, we can quantify how well our model learned from the training random samples. Generally speaking, the smaller the NLL is, the more information we capture from the training data set. Notice that the theoretical lower bound of the NLL is  $\ln(|\mathcal{T}|)$ . Thus, if the NLL is equal to  $\ln(|\mathcal{T}|)$ , it means the KL divergence is zero, indicating that the distribution of our model is exactly the same as empirical data distribution. That is, our model has exactly recorded the entire training set and is able to generate samples identical to training data with an equal probability assigned to each of the training samples.

In Fig. 3(a), we show the NLL of the training set as a function of the number of training patterns  $|\mathcal{T}|$ . The dashed line is the NLL's theoretical limit  $\ln(|\mathcal{T}|)$ . As we can see, the NLL converges to the theoretical limit when the maximum bond dimension  $D_{\max} \geq |\mathcal{T}|$ . In fact, a MPS or TTN with a bond dimension equal to  $D$  could analytically encode  $D$  images [40]. So when  $D_{\max} \geq |\mathcal{T}|$ , TTNs have enough expressive power to converge to the theoretical limit.

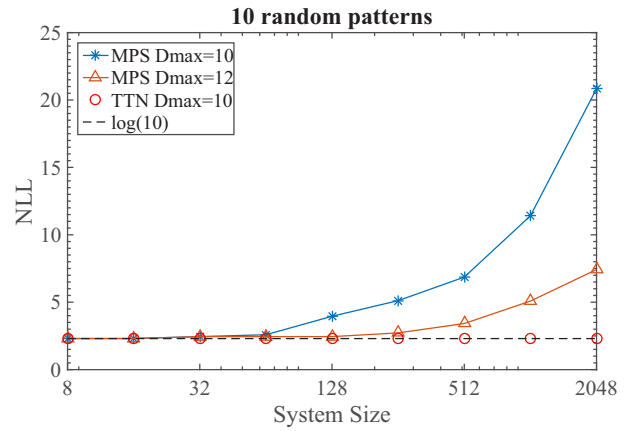


FIG. 4. Comparison between the TTN and MPS Born machines trained on ten random patterns with different system sizes. As the system size become larger, MPS can no longer reach the theoretical limit of the NLL when  $D_{\max}$  equals the number of samples, while the TTN is almost unaffected by the system size. This is because the structure of the TTN can capture the long-range dependences better.

As noticed in the traditional theory of tensor networks, the maximum entanglement entropy that a bond of the tensor network can capture equals  $\ln(D)$  [41]. For the random data sets considered here, the classical Shannon entropy of the TN approaches this value of entanglement entropy. But for more general cases the relation between entanglement entropy and classical Shannon entropy is not completely understood [42].

In Fig. 3(b) we plot the NLL as a function of the number of pixels in each random pattern. The number of training patterns  $|\mathcal{T}| = 50$ . Figure 3(b) shows that when  $|D_{\max}| < |\mathcal{T}|$ , the NLL increases almost linearly with the number of variables in the pattern. This is because the long-range correlations of a particular set of random patterns are dense, and the TTN does not have enough capacity to exactly record all the information of the random patterns. When  $|D_{\max}| \geq |\mathcal{T}|$ , since the correlation length of pixels in the TTN grows only logarithmically with the size of the image, the NLL can always easily converge to the theoretical limit regardless of how large the size of the picture is.

This point is further illustrated in Fig. 4, where the relationship between system size and training the NLL on different models is shown. As an example, we use  $|\mathcal{T}| = 10$  random patterns to train both the TTN and MPS models. We found that even at very large  $N$ , the TTN can still converge to the NLL's theoretical minimum once its maximum bond dimension reaches 10. However, under the same or even higher bond dimension ( $D_{\max} = 12$ ), the NLL of the MPS still fails in converging to the theoretical bound when the size is very large. Because in the MPS the correlation length decays exponentially fast, the information contained in the middle bond is more saturated when the image size becomes very large, making the maximum-likelihood training less efficient.

#### B. Binary MNIST data set

A standard benchmark for computer vision, especially for generative modeling, is the handwritten digits of the MNIST data set. The binary MNIST data set contains a training set

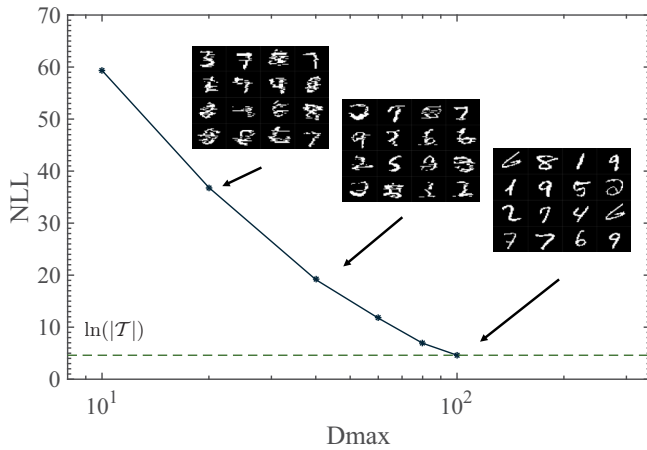


FIG. 5. Training the NLL and sampling images for a  $|\mathcal{T}| = 100$  binarized MNIST data set.  $\ln(|\mathcal{T}|)$  is the theoretical minimum of the NLL. The TTN exactly remembers all the information of the images when  $D_{\max} = |\mathcal{T}|$ .

of 50 000 images, a validation set of 10 000 images, and a test set of 10 000 images. Each of them contains handwritten digits of  $28 \times 28$  pixels with a value of 0 or 1. In order to facilitate comparison with other work, we directly use the same standard binary MNIST data set that has been used in the analysis of deep belief networks and has been widely recognized by the machine learning community [43]. The data set can be downloaded directly from the corresponding website [44].

We did three experiments on the binary MNIST data set. In the first experiment we used 100 randomly selected images to train TTNs with different  $D_{\max}$ . The results are shown in Fig. 5, where we can see that as the NLL gradually decreases, the quality of the generated samples becomes better. The training NLL will decrease to its theoretical minimum as  $D_{\max}$  increases to  $|\mathcal{T}|$  while the sampling image will be exactly the same as the one in the training set.

In Fig. 6 we plot the two-site correlation function of pixels. In each row, we randomly select three pixels, then calculate the correlation function of the selected pixels with all other pixels. The values of the correlations are represented by color. The real correlations extracted from the original data are illustrated in the top row, and correlations constructed from the learned MPS and TTN are shown in the bottom rows for comparison. For the TTN and MPS,  $D_{\max}$  is 50 and 100, respectively, which corresponds to the models with the smallest test NLL. As we can see, in the original data set, the correlation between pixels consists of short-range correlation and a small number of long-range correlations. However, the MPS model can faithfully represent the short-range correlation of pixels, while the TTN model performs well in both short-range and long-range correlations.

Next, we carried out experiments using the whole MNIST data set with 50 000 training images to quantitatively compare the performance of the TTN with existing popular machine learning models. The performance is characterized by evaluating the NLL on the 10 000 test images. We also applied the same data set to the tree-structure factor graph and the MPS generative model and compare using the same data set the test

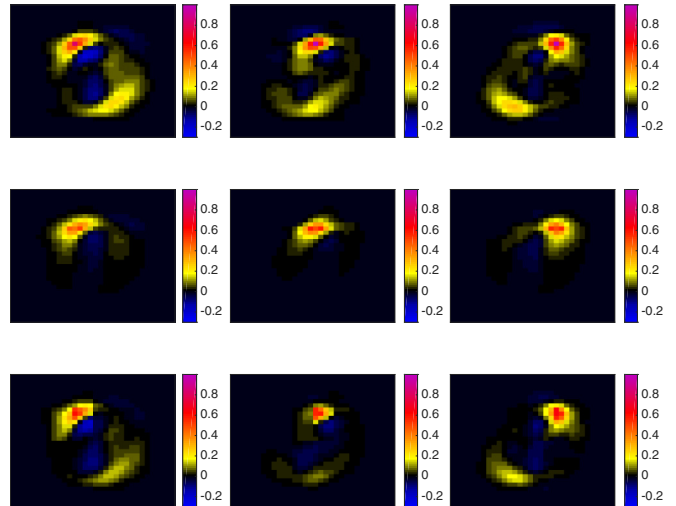


FIG. 6. Two-site correlation of pixels extracted from the original data (first row), the MPS (second row), and the TTN model (third row). We randomly choose three pixels at the tenth row of the images. The  $D_{\max}$  of the TTN is 50; the  $D_{\max}$  of the MPS is 100, which corresponds to the models with the smallest test NLL.

NLL with RBM, VAE, and PixelCNN, which currently gives the state-of-the-art performance. Among these results, RBM and VAE evaluate only approximately the partition function and hence give only an approximate NLL. However, the TTN and MPS together with PixelCNN are able to evaluate exactly the partition function and give exact NLL values.

The results are shown in Table I, where we can see that the test NLL obtained by the tree-structure factor graph is 175.8, and the result of the MPS is 101.45, with corresponding  $D_{\max} = 100$ . For the TTN in a 1D data representation [as depicted in Fig. 2(b)] with  $D_{\max} = 50$ , the test NLL already reduces to 96.88. With the same  $D_{\max}$ , the TTN performed on a 2D data representation [as depicted in Figs. 2(a) and 2(c)] can do even better, giving a NLL around 94.25. However, we see from Table I that when compared to the state-of-the-art machine learning models, the tensor network models still have a lot of space to improve: the RBM using 500 hidden neurons and 25-step contrastive divergence could reach a NLL of approximately 86.3, and PixelCNN with seven layers gives a NLL around 81.3.

In Fig. 7 we draw the sampled images from the TTN trained on 50 000 MNIST images, using the sampling

TABLE I. Test NLL of different models for the binary MNIST data set.

Model	Test NLL
Tree factor graph	175.8
MPS	101.5
TTN, 1D	96.9
TTN, 2D	94.3
RBM	86.3 <sup>a</sup> [43]
VAE	84.8 <sup>a</sup> [45]
PixelCNN	81.3 [10]

<sup>a</sup>Approximated NLL.

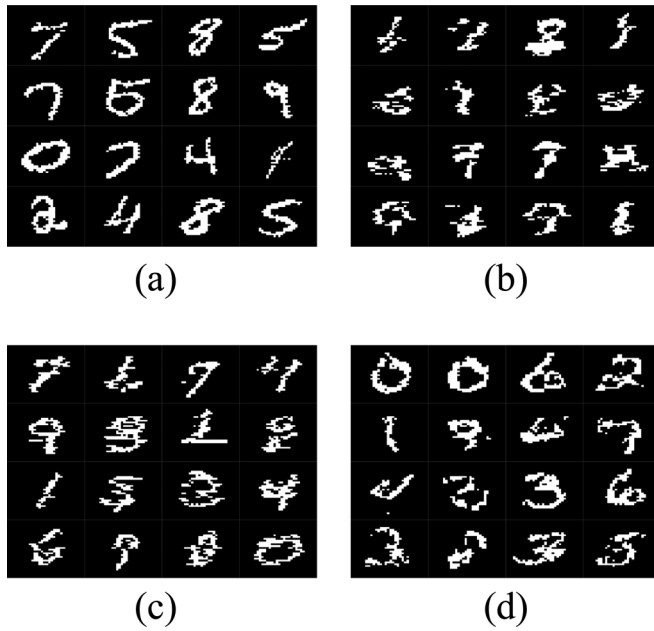


FIG. 7. Images generated by the MPS and TTN that are trained on  $|T| = 50\,000$  training images. (a) Part of the training images. (b) MPS with  $D_{\max} = 100$ , test NLL = 101.45, (c) 1D TTN with  $D_{\max} = 50$ , test NLL = 96.88, and (d) 2D TTN with  $D_{\max} = 50$ , test NLL = 94.25.

algorithm described in Sec. II F, and compare them with the images sampled from the MPS trained on the same data set. Figure 7 shows that TTNs with 2D data representation samples are visually better looking than MPS figures, indicating that the TTN captures global dependences better than the MPS.

#### IV. CONCLUSIONS AND DISCUSSION

We have presented a generative model based on tree tensor networks. This model is a direct extension of the matrix product state Born machine [14] and also a generalization of the tree-structure factor graph for generative modeling. The TTN inherits advantages of MPSs for generative modeling, including a tractable normalization factor, the canonical form, and direct sampling, but overcomes the issue of exponential decay of correlations in MPSs, making it more effective in capturing long-range correlations and perform better for large-size images. It is also straightforward to perform the TTN for two-dimensional modeling of images. We have developed efficient sweeping training algorithms for decreasing the NLL lost function using single-site and two-site updating schemes.

We have carried out extensive experiments to test the performance of the proposed TTN Born machine and compare it with existing approaches. We showed that the TTN gives a better training NLL than the MPS (with the same bond dimension) in remembering large random patterns. For classic

MNIST handwritten digits, the TTN captures long-range dependences better than the MPS and gives a much better NLL for test images, which indicates a better generalization power.

Naturally, further development of the current work would be to introduce the structure of the multiscale entanglement renormalization ansatz [46,47], another type of tensor network we can expect to have a tractable partition function while hopefully being able to preserve better the long-range dependences in the data.

We have also pointed out the gap between current generative models based on tensor networks and the state-of-the-art machine learning models based on neural networks such as PixelCNN. One advantage of neural-network-based models is the better prior for the images powered by the convolution. So an important step for tensor-network-based models to improve further is utilizing better priors of 2D images. Along with this direction, the PEPSs [22], which give a much better prior for natural images, should be considered. However, notice that this comes with the trade-off that the partition function is no longer exactly computable. This might not be a serious problem as approximate contraction algorithms such as the tensor renormalization group, boundary MPS, and corner transfer matrix have proved to be efficient in contracting PEPSs for finite-size systems. We will put this into future work.

We emphasize here that the necessity of developing generative learning algorithms based on tensor networks is mainly motivated by the quantum machine learning field. The machine learning model based on tensor network representation is essentially the type of model that uses a specific quantum state to represent classical data. Research on this type of model will pave the way for future migration of machine learning to quantum computers [25].

On the other hand, traditional machine learning models dealing with generative learning are closely linked to tensor networks dealing with quantum many-body problems. Finding the reason the machine learning model performs better than the tensor network for generative learning could also help the traditional tensor network algorithms continue to improve.

#### ACKNOWLEDGMENTS

We thank E. M. Stoudenmire, J. Chen, X. Gao, C. Peng, Z.-Y. Han, and J. Wang for inspiring discussions and collaborations. S.C. and T.X are supported by the National R&D Program of China (Grant No. 2017YFA0302901) and the National Natural Science Foundation of China (Grants No. 11190024 and No. 11474331). L.W. is supported by the Ministry of Science and Technology of China under Grant No. 2016YFA0300603 and the National Natural Science Foundation of China under Grant No. 11774398. P.Z. is supported by the Key Research Program of Frontier Sciences of CAS, Grant No. QYZDB-SSW-SYS032, and Project 11747601 of the National Natural Science Foundation of China.

[1] R. Salakhutdinov, Learning deep generative models, *Annu. Rev. Stat. Appl.* **2**, 361 (2015).

[2] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature (London)* **521**, 436 (2015).



- [3] J. Pearl, *Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning* (Computer Science Department, University of California, Los Angeles, 1985).
- [4] X. D. Huang, M. A. Jack, and Y. Ariki, *Hidden Markov Models for Speech Recognition* (Edinburgh University Press, Edinburgh, 1990).
- [5] N. Le Roux and Y. Bengio, Representational power of restricted Boltzmann machines and deep belief networks, *Neural Comput.* **20**, 1631 (2008).
- [6] G. E. Hinton, Deep belief networks, *Scholarpedia* **4**, 5947 (2009).
- [7] D. P. Kingma and M. Welling, Auto-encoding variational Bayes, [arXiv:1312.6114](https://arxiv.org/abs/1312.6114).
- [8] L. Dinh, J. Sohl-Dickstein, and S. Bengio, Density estimation using real NVP, [arXiv:1605.08803](https://arxiv.org/abs/1605.08803).
- [9] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, and A. Graves, Conditional image generation with PixelCNN decoders, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2016), pp. 4790–4798.
- [10] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, Pixel recurrent neural networks, [arXiv:1601.06759](https://arxiv.org/abs/1601.06759).
- [11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets, in *Advances in Neural Information Processing Systems 27*, edited by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Curran Associates, Inc., New York, NY, 2014), pp. 2672–2680.
- [12] G. E. Hinton and T. J. Sejnowski, Learning and relearning in Boltzmann machines, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (MIT Press, Cambridge, MA, 1986), Vol. 1, pp. 282–317.
- [13] S. Cheng, J. Chen, and L. Wang, Information perspective to probabilistic modeling: Boltzmann machines versus Born machines, *Entropy* **20**, 583 (2018).
- [14] Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, Unsupervised Generative Modeling Using Matrix Product States, *Phys. Rev. X* **8**, 031012 (2018).
- [15] J.-G. Liu and L. Wang, Differentiable learning of quantum circuit Born machines, *Phys. Rev. A* **98**, 062324 (2018).
- [16] Z. Li and P. Zhang, Shortcut matrix product states and its applications, [arXiv:1812.05248](https://arxiv.org/abs/1812.05248).
- [17] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, *Ann. Phys. (NY)* **349**, 117 (2014).
- [18] R. Orus, Advances on tensor network theory: Symmetries, fermions, entanglement, and holography, *Eur. Phys. J. B* **87**, 280 (2014).
- [19] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac, Matrix product state representations, [arXiv:quant-ph/0608197](https://arxiv.org/abs/quant-ph/0608197).
- [20] Y.-Y. Shi, L.-M. Duan, and G. Vidal, Classical simulation of quantum many-body systems with a tree tensor network, *Phys. Rev. A* **74**, 022320 (2006).
- [21] G. Vidal, Entanglement Renormalization, *Phys. Rev. Lett.* **99**, 220405 (2007).
- [22] F. Verstraete and J. I. Cirac, Renormalization algorithms for quantum-many body systems in two and higher dimensions, [arXiv:cond-mat/0407066](https://arxiv.org/abs/cond-mat/0407066).
- [23] Y. Levine, D. Yakira, N. Cohen, and A. Shashua, Deep learning and quantum entanglement: Fundamental connections with implications to network design, [arXiv:1704.01552](https://arxiv.org/abs/1704.01552).
- [24] C. Bény, Deep learning and the renormalization group, [arXiv:1301.3124](https://arxiv.org/abs/1301.3124).
- [25] X. Gao, Z. Zhang, and L. Duan, An efficient quantum algorithm for generative machine learning, [arXiv:1711.02038](https://arxiv.org/abs/1711.02038).
- [26] J. Eisert, M. Cramer, and M. B. Plenio, Colloquium: Area laws for the entanglement entropy, *Rev. Mod. Phys.* **82**, 277 (2010).
- [27] H. W. Lin, M. Tegmark, and D. Rolnick, Why does deep and cheap learning work so well? *J. Stat. Phys.* **168**, 1223 (2017).
- [28] P. Mehta and D. J. Schwab, An exact mapping between the variational renormalization group and deep learning, [arXiv:1410.3831](https://arxiv.org/abs/1410.3831).
- [29] V. Pestun and Y. Vlassopoulos, Tensor network language model, [arXiv:1710.10248](https://arxiv.org/abs/1710.10248).
- [30] D.-L. Deng, X. Li, and S. Das Sarma, Quantum Entanglement in Neural Network States, *Phys. Rev. X* **7**, 021021 (2017).
- [31] X. Gao and L.-M. Duan, Efficient representation of quantum many-body states with deep neural networks, *Nat. Commun.* **8**, 662 (2017).
- [32] J. Chen, S. Cheng, H. Xie, L. Wang, and T. Xiang, Equivalence of restricted Boltzmann machines and tensor network states, *Phys. Rev. B* **97**, 085104 (2018).
- [33] E. Stoudenmire and D. J. Schwab, Supervised learning with tensor networks, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2016), pp. 4799–4807.
- [34] D. Liu, S.-J. Ran, P. Wittek, C. Peng, R. Blázquez García, G. Su, and M. Lewenstein, Machine learning by two-dimensional hierarchical tensor networks: A quantum information theoretic perspective on deep architectures, [arXiv:1710.04833](https://arxiv.org/abs/1710.04833).
- [35] S. Kullback and R. A. Leibler, On information and sufficiency, *Ann. Math. Stat.* **22**, 79 (1951).
- [36] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, Factor graphs and the sum-product algorithm, *IEEE Trans. Inf. Theor.* **47**, 498 (2006).
- [37] M. Levin and C. P. Nave, Tensor Renormalization Group Approach to Two-Dimensional Classical Lattice Models, *Phys. Rev. Lett.* **99**, 120601 (2007).
- [38] L. Tagliacozzo, G. Evenbly, and G. Vidal, Simulation of two-dimensional quantum systems using a tree tensor network that exploits the entropic area law, *Phys. Rev. B* **80**, 235127 (2009).
- [39] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. USA* **79**, 2554 (1982).
- [40] C. Hubig, I. P. McCulloch, and U. Schollwöck, Generic construction of efficient matrix product operators, *Phys. Rev. B* **95**, 035129 (2017).
- [41] U. Schollwöck, The density-matrix renormalization group in the age of matrix product states, *Ann. Phys. (NY)* **326**, 96 (2011).
- [42] J.-M. Stéphan, S. Furukawa, G. Misguich, and V. Pasquier, Shannon and entanglement entropies of one- and two-dimensional critical wave functions, *Phys. Rev. B* **80**, 184421 (2009).
- [43] R. Salakhutdinov and I. Murray, On the quantitative analysis of deep belief networks, in *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*,

- edited by A. McCallum and S. Roweis (ACM, New York, NY, 2008), pp. 872–879.
- [44] [http://www.dmi.usherb.ca/~laroch/mlpython/\\_modules/datasets/binarized\\_mnist.html](http://www.dmi.usherb.ca/~laroch/mlpython/_modules/datasets/binarized_mnist.html)
- [45] Y. Burda, R. Grosse, and R. Salakhutdinov, Importance weighted autoencoders, [arXiv:1509.00519](https://arxiv.org/abs/1509.00519).
- [46] A. J. Ferris and G. Vidal, Perfect sampling with unitary tensor networks, *Phys. Rev. B* **85**, 165146 (2012).
- [47] A. J. Ferris and G. Vidal, Variational Monte Carlo with the multiscale entanglement renormalization ansatz, *Phys. Rev. B* **85**, 165147 (2012).