Differentiable programming of isometric tensor networks

IOP Publishing Mach. Learn.: Sci. Technol. 3 (2022) 015020 https://doi.org/10.1088/2632-2153/ac48a2

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
2 November 2021

REVISED
28 December 2021

ACCEPTED FOR PUBLICATION
6 January 2022

PUBLISHED
21 January 2022

PAPER

Differentiable programming of isometric tensor networks

Chenhua Geng[1,*], **Hong-Ye Hu**[2] and **Yijian Zou**[3]

[1] Institute for Solid State Physics, The University of Tokyo, Kashiwa, Chiba 277-8581, Japan
[2] Department of Physics, University of California San Diego, La Jolla, CA 92093, United States of America
[3] Stanford Institute for Theoretical Physics, Stanford University, Palo Alto, CA 94305, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: xwkgch@issp.u-tokyo.ac.jp

Keywords: tensor network, auto-differentiation, machine learning, condensed matter physics

Abstract

Differentiable programming is a new programming paradigm which enables large scale optimization through automatic calculation of gradients also known as auto-differentiation. This concept emerges from deep learning, and has also been generalized to tensor network optimizations. Here, we extend the differentiable programming to tensor networks with isometric constraints with applications to multiscale entanglement renormalization ansatz (MERA) and tensor network renormalization (TNR). By introducing several gradient-based optimization methods for the isometric tensor network and comparing with Evenbly–Vidal method, we show that auto-differentiation has a better performance for both stability and accuracy. We numerically tested our methods on 1D critical quantum Ising spin chain and 2D classical Ising model. We calculate the ground state energy for the 1D quantum model and internal energy for the classical model, and scaling dimensions of scaling operators and find they all agree with the theory well.

1. Introduction

Tensor network has been a powerful tool to study quantum many-body systems and classical statistical-mechanical models both theoretically [1, 2] and numerically [3–8]. And recently, it has been proposed as an alternative tool for (quantum) machine learning tasks, both for supervised learning [9–16], and unsupervised learning [17–22]. In the family of tensor networks, the multi-scale entanglement renormalization ansatz (MERA) and tensor network renormalization (TNR) are important tensor networks which are inspired by the idea of renormalization group (RG). They contain tensors that are to be optimized under isometric constraints, where some tensors are restricted to be isometric.

RG [23–25] plays an important role in modern condensed matter physics and high-energy physics. What lies in the heart of RG is the coarse graining procedure that changes the scale of the system. Under RG, microscopic models flow to different fixed points that distinguish different macroscopic phases of matter. More recently, the concept of RG also finds applications in machine learning and artificial intelligence (AI) [26–30]. Traditionally, RG is performed in the Fourier space, which involves various approximations. Furthermore, it has been shown that RG can be performed in real space using tensor networks.

In the context of quantum many-body physics, MERA and its variations [4, 31–34] are tensor networks which perform real-space RG on quantum states. Due to the specific structure, MERA is especially suitable for describing systems with scale invariance such as quantum critical systems. On the other hand, for a classical statistical mechanical model, TNR [34, 35] is a tensor network algorithm that performs real-space RG for the tensor network that represents the partition function of this model. Compared to tensor renormalization group (TRG) [36] which also performs real-space RG, TNR resolves the computational breakdown of TRG for critical systems. Additionally, it is known that TNR is closely related to MERA as formalized in [37].

From the perspective of theoretical physics, MERA and TNR can be used to extract universal information of critical systems, which are in a close relationship with conformal field theory (CFT) [38, 39]. Recently,

another variation of MERA using neural network was proposed and had been applied to simulation of quantum field theories, and finding holographic duality based on field theory actions [27, 40].

From the perspective of numerical studies, one of the central problems to tensor network is the optimization of tensors for various systems. Conventional optimization algorithms are designed manually and separately for different problems and different tensor networks structures. Recently, differentiable programming, as a novel programming paradigm, has been proposed for the optimization problems of tensor networks based on gradient optimization, which is extensively used in deep learning and artificial intelligence. Compared with traditional gradient estimation, such as finite difference method, automatic differentiation has the merits such as high accuracy and low computational complexity for calculating gradients for many parameters. It provides a unified and elegant solution to many optimization problems by combining the well-developed automatic differentiation frameworks like PyTorch [41] and TensorFlow [42]. Recently, it has been successfully applied to TRG and the optimization of projected entangled pair states (PEPS) [43]. With the help of differentiable programming researchers can focus on the part of tensor contraction in the algorithm without worrying about detailed and complicated optimization algorithms. However, tensor networks which possess isometric constraints, such as MERA and TNR cannot be optimized using differentiable programming directly.

Recently, Hauru *et al* [44] initiated the use of Riemannian optimization to tensor networks with isometric tensors. Specifically, they have applied preconditioned conjugate gradient method to MERA and matrix product states. In this work, we use a modified gradient search and show that auto-differentiation can be applied to MERA and TNR as well. We explicitly construct the computation graphs for MERA and TNR algorithms. Further, we discuss the gradient-based optimization methods and find that the combination of Evenbly–Vidal and gradient-based methods generally have a better performance than either of them. Taking 1D quantum and 2D classical Ising model as examples, we obtain the ground state energy for the 1D quantum model and internal energy for the 2D classical model with high accuracy. We also obtain scaling dimensions of scaling operators at the critical point within the differentiable programming framework.

This paper is organized as follows: in section 2 we review the idea of differentiable programming, including automatic differentiation, computation graphs and gradient-based optimization. We introduce our new methods for optimizing isometric tensors within differentiable programming framework. In sections 3 and 4 we briefly review MERA and TNR respectively, and show the results computed using differentiable programming. Finally we make a summary about differentiable programming and give our outlook for future research directions in section 5.

2. Differentiable programming

The characteristics of differentiable programming is automatic differentiation which computes the derivative information automatically by the well-developed frameworks. We first review the core concepts of differentiable programming and investigate how the derivative information are automatically computed for tensor networks. Then we discuss how to optimize a tensor network, especially with isometric constraints, by these derivative information.

2.1. Computation graphs

Computation graphs are central to the automatic differentiation which present how the derivatives are computed with respect to intermediate variables by the chain rule

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial X^n}\frac{\partial X^n}{\partial X^{n-1}}\cdots\frac{\partial X^1}{\partial \theta}, \tag{1}$$

where $\mathcal{L}$ is some general objective function used for the optimization, θ is a general trainable variable and X^i are intermediate variables. A computation graph is a directed acyclic graph where a node represents a tensor which stores the data and a link indicates the dependence of data flow during the computation process. The simplest computation graph is the chain graph characterized by equation (1), see figure 1.

In differentiable programming, all the data belongs to tensors which are represented by higher-order arrays. The data together with trainable variable θ flows along the computation graph with a series of intermediate tensors to obtain the final output results. This computation process is referred to as the forward propagation.

In most application scenarios, there are some tensors to be determined and optimized in the computation graph. For this purpose, the output results are usually compared with the expected output data to obtain a scalar, known as loss function, which evaluates the quality of the computation process. From the loss function, the derivatives with respect to the intermediate tensors are computed along the reversed computation graph, which is the so-called backward propagation. Once a computation graph has been set

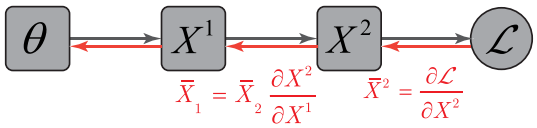

Figure 1. An example of the chain graph. The forward and backward propagation are indicated by black and red arrows.

up, one can directly obtain the derivative tensors of output with respect to the input or other intermediate tensors through backward propagation. Using the derivative information of the intermediate tensors, the parameters in the computation graph can be optimized by various optimization methods which will be discussed later.

For a general computation graph, the computation of the derivative of a node need to sum over all contributions from its child nodes

$$\bar{X}^i = \sum_{j:\text{child of } i} \bar{X}^j \frac{\partial X^j}{\partial X^i}, \tag{2}$$

where the adjoint variable $\bar{X} = \partial\mathcal{L}/\partial X$ is defined as the derivative of the final output $\mathcal{L}$ with respect to the variable X as shown in figure 1. If a node X has several different paths to flow and affect the final output, the derivative $\bar{X}$ will account all the contribution among these paths.

In the conventional tensor network computation, it usually involves computing the environment of a tensor which is also a tensor obtained by computing a fully contracted network without this tensor. These environment tensors are used to optimize the tensor network, but it is laborious to draw and determine the details of the contraction graph of the environment tensor by hands. Besides, because of the tensor contraction operation with a large amount of tensors, computing the environment of a tensor is costly for the computer. We note that the derivation with respect to a tensor is computed by removing the tensor and compute the contraction of the remnant tensor network, which is exactly the derivative of a tensor is exactly the environment of the tensor so long as the computation graph is constructed properly. The observation above motivates the application of automatic differentiation to optimizing tensor networks in order to get rid of the tedious environments computation. The differentiable programming can be easily implemented using modern machine learning frameworks, like PyTorch, TensorFlow. Furthermore, these frameworks provide easy-to-use interfaces with high performance computing units, such as graphics processing units (GPUs) and Tensor Processing Units (TPUs), which remarkably boost the computation speed.

2.2. Gradient-based optimization

Having established the computation graph, we need to determine the optimization method used in differentiable programming. Given the input data $\mathcal{D}$, the optimization problem is to find an optimal mapping $f(\mathcal{D})$ with parameters X to minimize the loss function

$$\min_X .\mathcal{L}(\mathcal{D}, f_X(\mathcal{D})). \tag{3}$$

Gradient descent method is the earliest and most common optimization method. The idea of gradient descent method is to iteratively update the parameter X by subtracting the gradient of the parameter $\bar{X}$ timed by a factor η, known as learning rate,

$$X_{t+1} \leftarrow X_t - \eta \bar{X}_t. \tag{4}$$

Some studies have found that in many cases the difficulty of optimization comes from the 'saddle point' [45] where the slope is positive in some directions and negative in another directions. There are several ways to help the optimization escape saddle points.

For example, the momentum [46] is introduced in gradient descent to simulate the inertia of optimization process. In the momentum method the historical influence is taken into account

$$M_{t+1} \leftarrow \beta_m M_t + \eta \bar{X}_t, \tag{5}$$

$$X_{t+1} \leftarrow X_t - M_{t+1}, \tag{6}$$

where $\beta_m \leqslant 1$ is the so-called momentum factor. With benefiting from the extra variable M, the momentum method can help speed up the convergence and get away from saddle points.

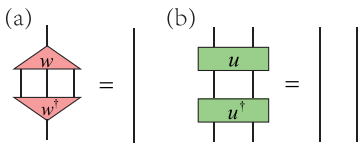

Figure 2. The isometric constraints for isometries w and disentanglers u.

Another direction of improving gradient descent is to adjust the learning rate dynamically. For instance, RMSprop [47] uses the historical decayed gradients accumulated up to adjust the learning rate automatically by

$$v_{t+1} \leftarrow \sqrt{\beta_v v_t + (1-\beta v)(\bar{X}_t)^2}, \tag{7}$$

$$X_{t+1} \leftarrow X_t - \frac{\eta}{v_{t+1}}\bar{X}_t, \tag{8}$$

where β_v is a decay factor.

2.3. Riemannian optimization for isometric tensors

In many physics problems, some of the tensors and parameters are required to be satisfied isometric constraints $X^TX = I$ where X is the parameter to be optimized of the form of matrices and tensors. The isometric constraints for tensor contraction is illustrated in figures 2 and 12. In this paper we assume the parameter X is real and X^T denotes the transpose of X. The generalization to complex cases is straightforward.

However, the isometric constraints, also referred as (semi-) orthogonal or (semi-) unitary constraints, obstruct the direct application of the optimization methods in the previous subsection. There are two ways to impose the constraints during the optimization.

One is the soft-constraint optimization, which allows the isometries and unitaries away from the constraints and add the deviation from the constraints into the loss function. This is known as the Lagrange multiplier method [48]. Then it becomes an unconstrained problem which the gradients of tensors are simply derivatives of tensors. We can directly use the inbuilt optimizers of the deep learning frameworks to minimize the modified loss function which is composed of the energy and deviation, see appendix F.

The other is the hard-constraint optimizations, which restrict the isometries and unitaries to be isometric. This corresponds to the optimization problems on the Stiefel manifold [49, 50]. In this case, the gradients of tensors should be confined in the Stiefel manifold, being different from simple derivatives of tensors which is the gradient tensors of the whole Euclid space. In the following, we focus on the hard-constraint optimization.

The hard-constraint optimization requires the tensors to be optimized satisfying the constraints during the whole computation process. A widely used strategy for solving the problem is the Riemannian optimization. Various methods based on Riemannian optimization have been studied extensively, including polar decomposition [51], QR decomposition [52], Cayley transform [50, 53–56]. The applications of Riemannian optimization have also been studied in physics like quantum control and quantum technologies [57, 58].

Basically, the Riemannian optimization includes two steps: (a) find the gradient vector in the tangent space of the current point on Stiefel manifold. (b) Find the descent direction and ensuring the new points on the manifold.

Specifically, assume the tensor X is one of the tensors to be optimized with the constraints. Define the set $\{X \in \mathbb{R}^{n\times p} : X^TX = I, n \geqslant p\}$ as the Stiefel manifold with the dimension $np - \frac{1}{2}p(p+1)$. And define the tangent space at the point X as $\mathcal{T}_X = \{Z \in \mathbb{R}^{n\times p} : Z^TX + X^TZ = 0\}$. Now the problem becomes to found a point on the Stiefel manifold with the minimum loss function $\mathcal{L}$

$$\min_{X\in\mathbb{R}^{n\times p}} .\mathcal{L}(X) \quad \text{s.t.}\, X^TX = I. \tag{9}$$

For the first step, the inner product in the tangent space $\mathcal{T}_X$ should be defined to obtain the gradient $G_X \in \mathcal{T}_X$. Let $Z_1, Z_2 \in \mathcal{T}_X$, we can define the Euclidean inner product as $\langle Z_1, Z_2 \rangle_e = \mathrm{tr}(Z_1^TZ_2)$. But it is more widely used to define a more natural choice which is the canonical inner product

$\langle Z_1, Z_2 \rangle_c = \mathrm{tr}(Z_1^T(I - \frac{1}{2}XX^T)Z_2)$. In the following we choose the canonical inner product. It can be proved [50] that the gradient G_X on the manifold is

$$G_X = AX = \bar{X} - \frac{1}{2}(XX^T\bar{X} + X\bar{X}^TX), \tag{10}$$

where $A = \bar{X}X^T - X\bar{X}^T + \frac{1}{2}X(\bar{X}^TX - X^T\bar{X})X^T$. In other words, the gradient G_X is obtained by projecting the derivative $\bar{X}$ onto the tangent space $\mathcal{T}_X$ of Stiefel manifold.

For the second step, the so-called operation retraction plays an important role. Generally speaking, there are two classes of retraction to keep the updated point on the manifold: projection-like and geodesic-like schemes. The projection-like schemes preserve the constraint by projecting a point into the manifold such as QR decomposition, while the geodesic-like schemes preserve the constraint by moving a point along the geodesic or quasi-geodesic line such as the Cayley transform.

2.3.1. Projection-like schemes

The point X on the manifold moves along the gradient vector a short distance to the new point $X - \eta G_X$, where η is a tunable parameter representing to the learning rate. But generally, the new point $X - \eta G_X$ is not the point on the manifold. We need to project the point $X - \eta G_X$ onto Stiefel manifold by various methods.

One way is the QR-decomposition-type retraction by noting that the Q factor of QR decomposition is orthogonal. Then the update equation is

$$\mathrm{QR} = X_t - \frac{1}{2}\eta G_X, \tag{11}$$

$$X_{t+1} = Q. \tag{12}$$

Another way is to use SVD as retraction. If the SVD of a matrix $X \in \mathbb{R}^{n\times p}$ is $X = U\Sigma V^T$, then the projection map onto Stiefel manifold is $\pi(X) = UI_{n\times p}V^T$ [59]. Then the update equation is

$$U\Sigma V^T = X_t - \eta G_X, \tag{13}$$

$$X_{t+1} = UI_{n\times p}V^T. \tag{14}$$

2.3.2. Geodesic-like schemes

The point X on the manifold moves along a single parameter curve $Y(\eta)$ such that the curve is on the Stiefel manifold, i.e. $Y(\eta)^TY(\eta) = I$, and the derivative of the curve at origin is the gradient vector $Y'(0) = -G_X$.

One of the choice of the curve is Cayley transform

$$Y(\eta) = \left(I + \frac{\eta}{2}A\right)^{-1}\left(I - \frac{\eta}{2}A\right)X, \tag{15}$$

where A is the matrix in equation (10). Then update equation is

$$X_{t+1} = \left(I + \frac{\eta}{2}A_t\right)^{-1}\left(I - \frac{\eta}{2}A_t\right)X_t. \tag{16}$$

Note that the inverting a $n \times n$ matrix $I + \frac{\eta}{2}A$ is costly. In appendix B, we show that the computation expense can be reduced by Sherman–Morrison–Woodbury formula or an iteration method.

2.3.2.1. Algorithm improvement

In order to improve the optimization performance, We apply the optimization techniques introduced in section 2.2 to our Riemannian optimization methods. Generally, both the accuracy and optimization speed could be improved by introducing the dynamic momentum and adaptive learning rate techniques [56].

To be specific, instead of directly using the derivative $\bar{X}$ in the gradient computation equation (10), we introduce a decorated matrix momentum M to replace $\bar{X}$. Then the momentum and the gradient are computed by

$$M_{t+1} \leftarrow \beta_m M_t + \bar{X}_t, \tag{17}$$

$$G_X \leftarrow M_{t+1} - \frac{1}{2}(X_tX_t^TM_{t+1} + X_tM_{t+1}^TX_t), \tag{18}$$

$$M_{t+1} \leftarrow G_X, \tag{19}$$

$$X_{t+1} \leftarrow R_{\eta}^{G_X}(X_t), \tag{20}$$

where $M_0 = 0$ and β_m is a hyperparameter to be tuned. Here $R_{\eta}^{G_X}(X_t)$ denotes the retraction operation, which can be substituted by equations (11)–(14) and (16). Note that we project the momentum onto the tangent space of Stiefel manifold.

A variant adaptive learning rate is also utilized by

$$\eta_{\text{adapt}} = \min.(\eta, \alpha_\eta/(\|A\| + \epsilon)), \tag{21}$$

where α_η is a hyperparameter to be tuned and $\|A\|$ denotes the norm of tensor A. The adaptive learning rate is such that a large learning rate is used when the norm of gradient is small. These techniques may improve the gradient diffusion problem during the optimization process and accelerate the convergence speed.

3. Application to MERA

The MERA is a variational ansatz for the ground state and low-energy excited states of critical quantum spin chains. Numerically, it has been used to extract universal information, such as scaling dimensions and operator product coefficients from the critical spin chain Hamiltonian. Furthermore, it has found applications in the context of holography [60] and emergent geometry [61].

3.1. Review of MERA

The MERA $|\psi(u,w)\rangle$ is composed of layers of isometries w and disentanglers u which satisfy the isometric constraints,

$$u^\dagger u = uu^\dagger = I, \quad ww^\dagger = I, \tag{22}$$

which are expressed graphically in figure 2. Physically, the MERA can be viewed as a renormalization group flow in the real space. From the bottom upwards, each layer of MERA coarse-grains the quantum state. The isometric constraints of the tensors are essential from both physical and numerical perspectives. From the physical perspective, the isometric constraints keep the norm of the state and retain the causal structure [32] under the coarse graining. From the numerical perspective, the isometric constraints of the tensors ensures that expectation values of the local operators can be computed in polynomial time.

Depending on the number of bonds of the isometries w, there are different types of MERA [62] and in this work we focus on the ternary MERA as in figure 3. We further assume translational invariance, where the u and w tensors are the same in the same layer. We also assume scale invariance, where all u's and w's are the same above a certain number of transitional layers. Let the number of transitional layers to be n, then the variational ansatz is completely specified by u_τ and w_τ, where $\tau = 1, 2, \ldots, n+1$ denotes the layers from bottom to top, and u_{n+1} and w_{n+1} constitute the scale-invariant layers.

In order to obtain the ground state, one optimizes the energy function as the loss function,

$$E = \frac{\langle\psi(u,w)|H|\psi(u,w)\rangle}{\langle\psi(u,w)|\psi(u,w)\rangle} = \langle\psi(u,w)|H|\psi(u,w)\rangle, \tag{23}$$

where in the second equality we have used the normalization of the state as a result of the isometric constraints.

It is well known [31] that the computation of the expectation value of local operators in a MERA involves two superoperators, the 'ascending superoperator' and the 'descending superoperator'. The ascending superoperator $\bar{\mathcal{A}}$ transforms local operators $o_{\tau-1}$ at layer $\tau - 1$ to local operators o_τ at layer τ, $o_\tau = \bar{\mathcal{A}}_\tau(o_{\tau-1})$, as shown in figure 4. The descending superoperator, as the adjoint of the ascending superoperator, transforms the two-site reduced density matrix ρ_τ at layer τ to the two-site reduced density matrix $\rho_{\tau-1}$ at layer τ, $\rho_\tau = \bar{\mathcal{D}}_\tau(\rho_{\tau+1})$, as shown in figure 5.

The coarse-graining is such that expectation values of the operator o_τ for each layer τ are the same

$$\text{tr}(o\rho) = \text{tr}(o_\tau \rho_\tau). \tag{24}$$

At the scale invariant layer $T \equiv n+1$, the two-site reduced density matrix satisfies

$$\rho_T = \bar{\mathcal{D}}_T(\rho_T), \tag{25}$$

which is the unique eigenoperator of the average ascending superoperator $\bar{\mathcal{D}}_T$ with eigenvalue 1. The eigenoperator can be approximately computed by the power method, i.e. repeatedly applying $\bar{\mathcal{D}}_T$ to any initial state until convergent.

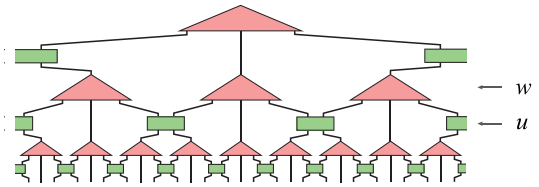

Figure 3. A one-dimensional infinite ternary MERA with isometries w and disentanglers u.

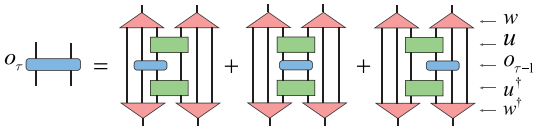

Figure 4. Ascending superoperators ascend the operator $o_{\tau-1}$ at the layer $\tau-1$ to the operator o_τ at the layer τ. They include three different form $\mathcal{A}_L$, $\mathcal{A}_C$ and $\mathcal{A}_R$. The average ascending superoperators $\bar{\mathcal{A}} = (\mathcal{A}_L + \mathcal{A}_C + \mathcal{A}_R)/3$.

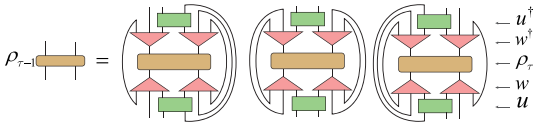

Figure 5. Descending superoperators descend the density tensor ρ_τ at the layer τ to the density tensor $\rho_{\tau-1}$ at the layer $\tau-1$. They include three different form $\mathcal{D}_L$, $\mathcal{D}_C$ and $\mathcal{D}_R$. The average descending superoperators $\bar{\mathcal{D}} = (\mathcal{D}_L + \mathcal{D}_C + \mathcal{D}_R)/3$.

To summarize, starting with random isometric tensors w, u under the constraints equation (22), the standard computation process of constructing a MERA involves two steps [31]:

(a) Top-bottom: computing the density tensor ρ_T at the top layer by solving the eigenoperator of $\bar{\mathcal{D}}_T$, then computing the density tensors ρ_τ for all layers $\tau < T$ from top to bottom by the descending superoperators $\bar{\mathcal{D}}_\tau$.
(b) Bottom-Top: From bottom to top, update the isometry and disentangler tensors w_τ and u_τ, and compute the Hamiltonian H_τ for all layers τ by ascending superoperators $\bar{\mathcal{A}}_\tau$.

For the updating procedure, we can use Evenbly–Vidal method [31] (see appendix A) or gradient-based methods introduced above. Repeating the (a) top-bottom and (b) bottom-top process, the tensors w and u in MERA will be constructed.

We note that in differentiable programming only the top-bottom approach is used and the bottom-top approach is computed automatically once the optimization method is determined.

3.1.1. Scaling dimensions

Once the MERA has been constructed and optimized, we can easily extract the scaling dimensions, which are universal properties of the phase transition that we can obtain easily from MERA. Take our ternary MERA as example, consider an on-site operator ϕ_α in scaling invariant layers. By the on-site ascending superoperator figure 6, the operator ϕ_α is lifted to $\bar{\mathcal{A}}_\tau(\phi_\alpha)$ in the next scale. The scaling operators are found by the fixed points of the ascending superoperator.

$$\bar{\mathcal{A}}_\tau(\phi_\alpha) = \lambda_\alpha \phi_\alpha. \tag{26}$$

It can be easily shown that the two-point correlator of such scaling operators are

$$\langle \phi_\alpha(3r)\phi_\alpha(0)\rangle = \lambda_\alpha^2 \langle \phi_\alpha(r)\phi_\alpha(0)\rangle. \tag{27}$$

The scaling dimensions can therefore be calculated by $\Delta_\alpha = -\log_3 \lambda_\alpha$.

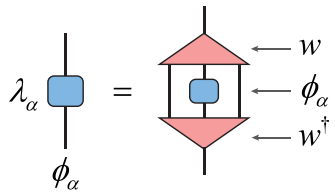

Figure 6. The on-site ascending superoperator acts on the eigenstate ϕ_α with the eigenvalue λ_α.

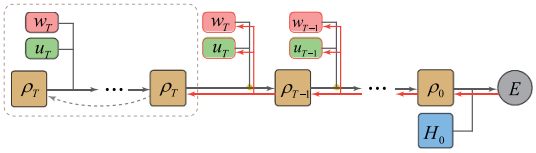

Figure 7. The computation graph for translation invariant MERA. At the left part the density tensor is iterated several times in the scaling invariant layer as the input data. The forward propagation (grey arrow) involves the descending operation with parameters w and u as in figure 5. On the bottom layer, the density matrix is contracted with the Hamiltonian to obtain the energy as the loss function. The backward propagation (red arrow) computes the derivative tensors $\bar{w}_\tau$, $\bar{u}_\tau$ and $\bar{\rho}_\tau$ automatically. Finally the parameters w and u are updated by the gradient optimization method.

3.2. Auto differentiation

3.2.1. Computation graph

The computation graph for the loss function of the optimization of MERA is shown in figure 7. The forward and backward propagation correspond to the top-bottom and bottom-top processes in the conventional computation, respectively. The top layer reduced density matrix ρ_T is computed by iteration of the descending superoperator for several layers. The isometries w and disentanglers u serve as network parameters to be trained. For each layer τ, the reduced density matrix ρ_τ flows to that of the lower layer $\rho_{\tau-1}$ by the descending superoperators. Note that the descending superoperators only involve tensor contractions which can be backward propagated automatically. At the bottom layer, the density matrix ρ_0 is contracted with Hamiltonian H_0 to obtain the energy E as the loss function $\mathcal{L}$. Then the derivative tensors $\bar{w}_\tau$, $\bar{u}_\tau$ and $\bar{\rho}_\tau$ of each layer τ are computed during automatically by backward propagation. We note that $\bar{w}_\tau$ and $\bar{u}_\tau$ equal to the environment tensors of w and u, respectively. Finally, due to equation (24), we have

$$\bar{\rho}_\tau = \frac{\partial E}{\partial \rho_\tau} = \frac{\partial \mathrm{tr}(H_\tau \rho_\tau)}{\partial \rho_\tau} = H_\tau. \tag{28}$$

The optimization of w_τ and u_τ can be done by the gradient optimization as in section 2. This can be combined with the traditional Evenbly–Vidal algorithm [31] and we compare the performance below.

3.2.2. Results

We use the critical one dimensional transverse field Ising model [63] to test the algorithm.

$$H_0 = -\sum_r \left(\sigma_x^{[r]}\sigma_x^{[r+1]} + \lambda\sigma_z^{[r]}\right), \tag{29}$$

where $\lambda = 1$ for the critical point.

The ground state energy density is known exactly,

$$E_{\mathrm{exact}} = -\frac{1}{2\pi}\int_{-\pi}^{\pi}\sqrt{(1-\cos k)^2 + \sin^2 k}\,dk = -\frac{4}{\pi}. \tag{30}$$

We use the PyTorch framework in Python to realize the auto-differentiable algorithm with the GPU acceleration. As comparison of the computation speed, for the optimization of a MERA with $\chi = 8$ in 10^4 iterations using Evenbly–Vidal method, our differentiable programming with GPU acceleration costs 883 s, while the conventional programming without GPU acceleration costs 5109 s. For gradient-based methods the GPU can also accelerate the computation speed. Our test platform is a laptop with Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz and NVIDIA GeForce GTX 1060 GPU.

In addition to the Riemannian optimization methods mentioned in 2.3, we also propose a random mixed method combining the Evenbly–Vidal method and the gradient-based methods. In random mixed method,

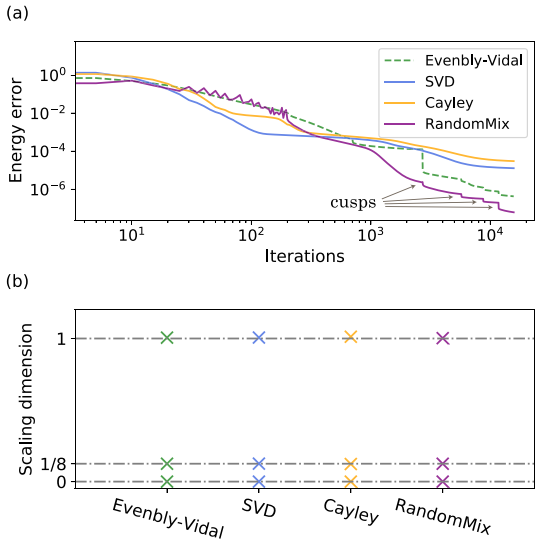

Figure 8. (a) The comparison of optimization curves with Evenbly–Vidal method (green dashed line), SVD method (blue solid line), Cayley method (yellow solid line) and the random mixed method (purple solid line). The max bond dimension χ is 12 and the number of transitional layers is 3. (b) The computed scaling dimensions with the four method comparing with the exact values (grey dot-dashed line).

we use Evenbly–Vidal method as basis method for iterations and replace an iteration by gradient descent methods with SVD or Cayley retractions every five iterations.

The error in energy for Evenbly–Vidal, SVD, Cayley and the random mixed methods are shown in figure 8(a). For SVD and Cayley methods, we introduce dynamical momentum and adaptive learning rate techniques with $\eta = 1.0$, $\beta_m = 0.9$ and $\alpha_\eta = 4.0$. The learning rate is decayed every ten iterations with the decay factor 0.999. Benefiting from the momentum and adaptive learning rate techniques, the errors of energy can be reduced by more than an order of magnitude.

It is known that the MERA optimization is vulnerable to be stuck into local minima and gradient diffusion [44]. In the practice of Ising model optimization, we find that if after the first several hundred iterations the energy error is still larger than a threshold, it will be high probability that the optimization is stuck into local minima. In order to reduce the chance of being stuck into local minima and improve the success rate of optimization, we used a resetting mechanism, see appendix C for details of resetting mechanism. The resetting here means that the network parameters w and u and the two-site reduced density matrix are set to the initial state. The number of resetting times can be used as an indicator to show the stability of optimization. High stability means that the optimization has a low possibility to be stuck at a local minima. We can see from figure 8(a) that for gradient-based methods (SVD, Cayley and the random mixed methods) the energy errors fall quickly at the beginning, meaning that the possibility of being stuck into local minima is lower than the Evenbly–Vidal method. Indeed, we find that these methods hardly trigger the resetting mechanism, while the Evenbly–Vidal method has certain possibility to trigger the resetting mechanism.

In order to speed up convergence, we applied a gradually lifting bond dimension trick with the bond dimension χ increasing from 4 to 12, see appendix C for details of lifting bond dimension trick. We can see in figures 8(a) and 9 that there are some cusps in the errors of energy, corresponding to the bond dimension lifting.

In figure 8(a), we find that although the accuracies of SVD and Cayley methods are not good as Evenbly–Vidal method, the combination of them with Evenbly–Vidal method, corresponding to the random mixed method, has a better accuracy. We repeat the same optimization process several times for Evenbly–Vidal method and random mixed method as shown in figure 9, finding that generally the random mixed method has better performance in both stability and accuracy. For other models the performance can also be improved by applying random mixed method, see appendix D.

We can provide an explanation for the performance improvement. It is known that the main difficulty of the optimization comes from the saddle points [45]. One idea for escaping saddle points is to introduce fluctuation such as using random input data in Stochastic gradient descent (SGD) [64]. However, in our problem the input data is deterministic (determined by w and u of the top layer in MERA). Therefore we introduce perturbation of the optimization with Evenbly–Vidal method by randomly choosing SVD and Cayley methods and adding an iteration with this method into the optimization process, as what did in the

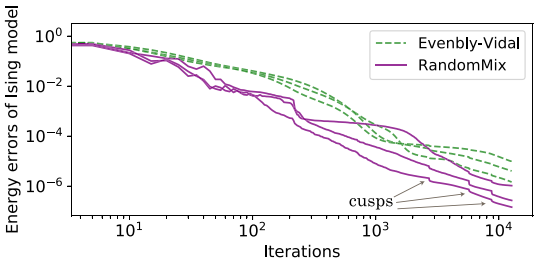

Figure 9. Repeated energy errors with Evenbly–Vidal and random mixed methods with the same setting as figure 8, except for the max bond dimension $\chi = 10$ here.

random mixed method. Different methods possess different searching way in the parameter space. Switching the method meaning the changing of searching way, which can help the optimization process to escape saddle points. As a result, such a combination of different optimization methods can improve both stability and accuracy benefiting from 'shaking up the system'.

The scaling dimensions can also be computed using the optimized MERA. The first three scaling dimensions using the MERA optimized by Evenbly–Vidal, SVD, Cayley and the random-mixed methods are shown in figure 8(b). As we can see, our optimized MERA could produce good value of scaling dimensions for the first several scaling dimensions. The scaling dimensions of higher order terms usually needs tensor networks with larger bond dimensions.

4. Application to TNR

The partition function of a $d+1$-dimensional classical statistical-mechanical models or d-dimensional quantum many-body systems can be represented as a $d+1$-dimensional tensor network. One way of computing the partition function is based on the real-space RG, where the linear size of the tensor network is reduced at each step. There are several methods that stood out, including the TRG [36], HOTRG [65], TNR [34, 35], Loop-TNR [66] and Gilt-TNR [67]. Most of the techniques work extremely well in $d = 1$, and some of them also work in $d = 2$. In this work we focus on TNR in $d = 1$, which produces a proper RG flow for a critical system. In TNR, there is an optimization over disentanglers and isometries similar to MERA [37]. Therefore, the differentiable programming techniques in this work can be applied. This generalizes previous work where the application of differentiable programming to TRG is discussed [43].

4.1. Review of TNR

Here we briefly review the algorithm of TNR. For more details we refer to [35]. Consider the 2D classical Ising model on square lattice with inverse temperature β. The partition function is

$$Z = \sum_{\{\sigma\}} e^{-\beta H(\sigma)}, \tag{31}$$

where

$$H(\sigma) = -\sum_{\langle i,j\rangle} \sigma_i \sigma_j, \tag{32}$$

and $\sigma_i = \pm 1$ is the Ising spin on site i. The partition function is a square tensor network consisting of four-index tensors A_{ijkl} which are located in the center of every second plaquette of Ising spins as in figure 10, where

$$A_{ijkl} = e^{\beta(\sigma_i\sigma_j + \sigma_j\sigma_k + \sigma_k\sigma_l + \sigma_l\sigma_i)}. \tag{33}$$

Then the patition function equation (31) is given by the contraction of the tensor network

$$Z(\beta) = \sum_{ijk\cdots} A_{ijkl} A_{mnoj} A_{krst} \cdots. \tag{34}$$

At each step of the coarse-graining, we approximate a 2×2 block of tensors by inserting isometries and unitaries as in figure 11, where the tensors v_L, v_R, u satisfy the isometric constraints as in figures 12(a)–(c). The tensors v_L, v_R, u are determined by minimizing the approximation error as in figure 13.

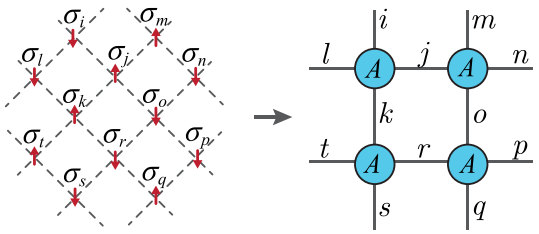

Figure 10. The square lattice of classical spins $\sigma = \pm 1$ rotated by 45° maps into a tensor network.

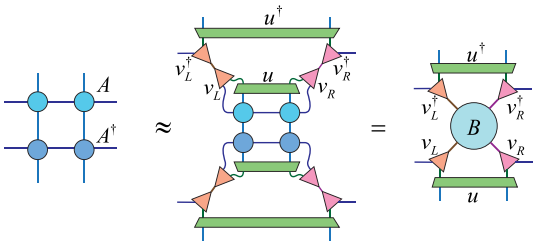

Figure 11. The original sub-network is approximated by a new network by inserting pairs of isometries and disentanglers. The tensor B is defined for convenience.

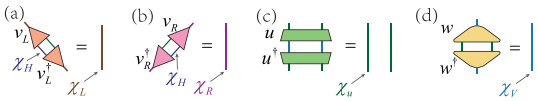

Figure 12. The isometric constraints for (a) the isometries v_L, (b) the isometries v_R, (c) the disentanglers u and (d) the isometries w. The bond dimension χ of each index are illustrated.

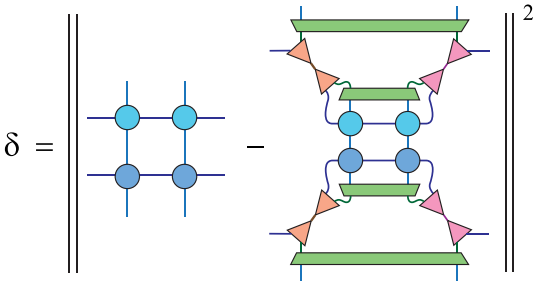

Figure 13. The loss function defined as the approximation error δ.

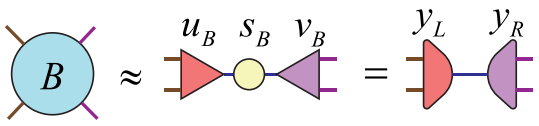

Figure 14. The approximation of tensor B by making singular value decomposition.

The optimization has been achieved by Evenbly–Vidal method as in [35]. Here we use the gradient-based methods like SVD and Cayley methods discussed in this paper.

Then, we contract some of the tensors into a tensor B as defined in figure 11. Next, a singular value decomposition on the tensor B is performed to obtain the isometric tensors u_B, v_B and the diagonal matrix s_B containing singular values, which is truncated to $\chi \times \chi$ by discarding smallest diagonals. The diagonal matrix s_B is then splited to make a pair of $\sqrt{s_B}$, and we define new tensors $y_L = u_B\sqrt{s_B}$ and $y_R = \sqrt{s_B}v_B$ as shown in figure 14. The tensor network is now entirely composed of the tensor on the left of figure 15.

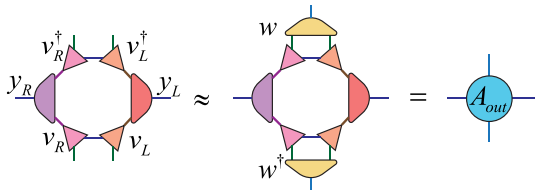

Figure 15. The definition of the coarse grained tensor A_{out}.

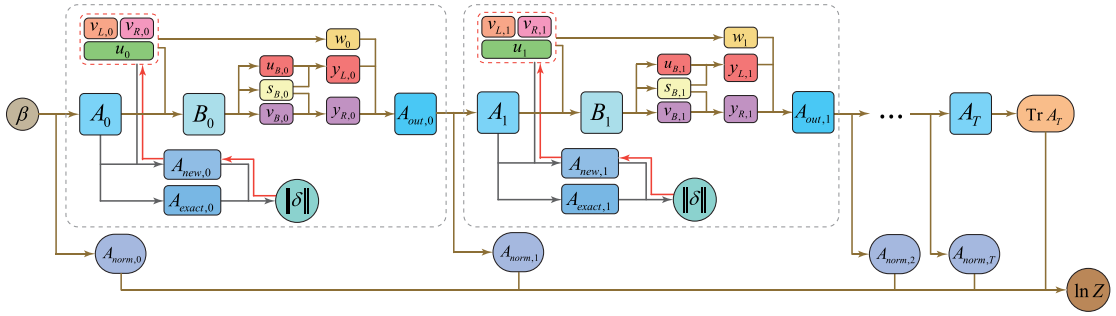

Figure 16. The computation graph for TNR. The grey dotted rounded rectangle indicate one step of TNR computation. The forward and backward propagation processes appear in each coarse-graining step. The gray arrows show the forward propagation path, while the red arrows show the backward propagation path.

Finally, we insert the isometry w with the isometric constraint shown in figure 12(d) and obtain A_{out}, which effectively contain the information of four tensors in the original tensor network.

We note that the tensors w should be also optimized in order to minimize the approximation errors as in figure 13. But here we can apply an alternative method by solving the eigenvectors of the matrix of the first sub-network of figure 15 with contracting the left and right indices, because of the hermitian property of this sub network.

Repeating the computation above, the original large tensor network of the partition function can be simplified to one or a few tensors, which can be easily computed. Also note that at each RG step, the tensor A is divided by the norm of A to prevent the data explosion.

4.2. Auto differentiation

4.2.1. Computation graph

Within the differentiable programming framework, we show the computation graph of TNR in figure 16.

Given the inverse temperature β, we first construct the tensor network representation for the system. Then we use TNR to coarse grain the tensor network until the network only consists of single or a few tensors. At each RG step, we use the truncation error as the loss function, and then backward propagate to optimize the parameters v_L, v_R and u. We refer to appendix E for detail description of the computation graph of TNR.

Since the computation graph for the loss function is short, the optimization of parameters v_L, v_R and u is easy with the fast convergence. At the last step where only one tensor A_T remains in the network, we obtain the partition function $\ln Z$ by taking the tensor trace of A_T and multiplied by the A_{norm} of all previous steps.

4.2.2. Results

For 2D classical Ising model on an infinite square lattice, the logarithm of partition function per site is exactly known [68]

$$\ln Z = \frac{1}{2}\ln 2 + \frac{1}{2\pi}\int_0^\pi \ln\left(\cosh^2 2\beta + \frac{1}{\lambda}\sqrt{1+\lambda^2-2\lambda\cos 2x}\right)dx, \tag{35}$$

where $\lambda = 1/\sinh\beta^2$.

In figure 17(a), we show the computed $\ln Z$ accords well with the exact value as a function of the inverse temperature β by random mixed method. We can see that the computed results by differentiable programming fit well with the exact values.

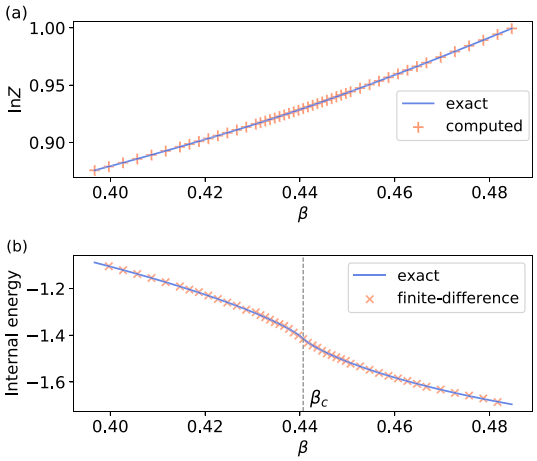

Figure 17. (a) The $\ln Z$ of 2D classical Ising model computed by differentiable programming with respected to the inverse temperature β. The blue line represents the exact value while the orange cross points represent the computed results by differentiable programming. The size of system is $2^8 \times 2^8$ and the max bond dimension here is $\chi = 14$. (b) The internal energy as the function of β computed by finite difference method.

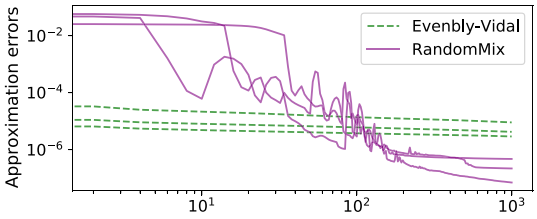

Figure 18. The approximation errors of 2D classical Ising model computed by differentiable programming with Evenbly–Vidal method and random mixed method. Here we take approximation errors of the 3rd, 5th and 7th layers as examples. The bond dimension here is $\chi = 14$.

We show the computed internal energy results $E = -\frac{\partial \ln Z}{\partial \beta}$ in figure 17(b) by finite difference method. The relative errors of $\ln Z$ and internal energy E are about $10^{-7} \sim 10^{-5}$ and $10^{-5} \sim 10^{-3}$ correspondingly. The errors are almost the same for the results from conventional TNR algorithm. But the computation speed can be boosted by GPU acceleration. We find the random mixed method can also improve the performance of optimization in TNR, as shown in figure 18.

As a comparison of computation speed, for the construction of a $2^8 \times 2^8$ TNR with $\chi = 14$ with 800 iterations for each layer using random mixed method, our differentiable programming with GPU acceleration costs 708 s, while the conventional programming without GPU acceleration costs 2456 s.

We note that in our cases, the automatic differentiation of quantities like internal energy $E = -\frac{\partial \ln Z}{\partial \beta}$ is difficult. Taking TNR as example, the loss function is a function of β. Each iteration of updating a parameter tensor will introduce a term of β. When we intend to automatic differentiate $\ln Z$ with respected to β, we should track all the hundreds and thousands of iterations and back-forward propagate the whole computation process. This make the automatic differentiation with respected to β difficult both for programming and efficient. For programming, it requires to build the computation graph of backward propagation itself. In other words, it need to backward propagate the backward propagation. But it is quite difficult to code such programming within automatic differentiation frameworks. For efficiency, we have to record and store the whole computation process with hundreds and thousands of iterations and track them back to the initial status. This procedure will consume a huge amount of memory resources as well as time.

4.2.3. Scaling dimensions

Once the TNR has been constructed and optimized, the scaling dimensions can also be extracted easily. Here we use the transfer matrix technique [69] to compute the scaling dimensions with the critical inverse temperature $\beta_c = \ln(1+\sqrt{2})/2$ shown in figure 19. We can see that with the TNR and transfer matrix technique the computed scaling dimensions have accurate results even at quite higher orders.

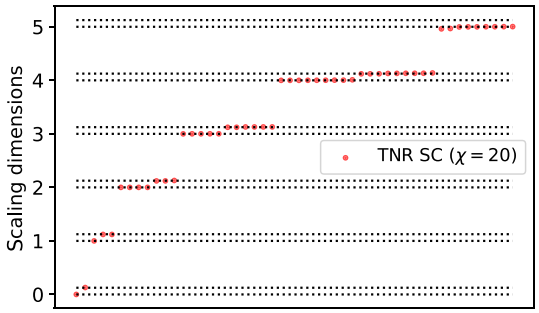

Figure 19. The scaling dimensions of 2D classical Ising model computed by differentiable programming. The dotted gray lines indicate the exact values. The size of system is $2^{10} \times 2^{10}$ and the max bond dimension here is $\chi = 20$.

5. Summary

In this paper, we extend the application of differentiable programming to an important family of tensor networks with isometric constraints such as MERA and TNR. We show how the MERA and TNR are constructed and optimized within differentiable programming framework by explicitly illustrating the computation graphs of them. We discuss several gradient-based optimization methods that supplement traditional methods for tensor networks with isometric constraints and show that the performance of optimization can be improved by combining the gradient-based method and traditional methods.

The differentiable programming in tensor network optimization has several advantages. (a) Differentiable programming has an unified programming paradigm since we only need to specify the computation graph and the optimization method for the optimization. (b) In differentiable programming, the gradient can be computed automatically, which releases the labor on the tedious contraction computation of environment graphs. (c) Differentiable programming has extensibility and flexibility in practice. It can be modularized, in which one can easily change and add different network layers and optimization methods. (d) We can also benefit from the parallel computation with the GPU acceleration. (e) Last but not the least, differentiable programming can also help with the optimization of arbitrary network networks when no traditional optimization methods are known.

With the merits above, the differentiable programming can be applied to other tensor networks such as higher-dimensional MERA or TNR, which have not yet been implemented previously due to the complexity of the network structures and the high computation expenses. With the help of automatic differentiation, we can get rid of the tedious labor of contracting graphs. For example, in our one dimensional ternary MERA case, we need only determine 3 tensor contraction graphs (descending superoperators) in differentiable programming, comparing with the 15 tensor contraction graphs in the traditional algorithm. This advantage will play a more significant role when the network structure becomes complicated. It is also very interesting and challenging to combine the optimization of tensor data together with the optimization of tensor network's structure [70]. This will serve as a unified method to discover hidden structures of the data. For example, given the measurement results of a quantum state, can machine find the best tensor network structure that represents the data? It has been shown that based on the entanglement data, neural network can establish different hidden geometry (structure) for area law, logarithmic law, and volume law quantum states as an analog of holographic duality [71]. In tensor network studies, it is also known that random tensor networks are closely related to holographic duality [72]. Whether different hidden geometry of tensor network can emerge purely based on observation data of different quantum states is both an interesting and fundamental question for physics. Our methods can also be applied to simulate and train quantum neural network ansatz for real-world applications [30]. Last but not the least, tensor networks with unitary or isometric constraints are closely related to quantum computing simulation and quantum machine learning [73], as unitary tensors are represented by quantum gates. The isometric tensor network representation is therefore a natural description and potential platform for quantum and quantum–classical hybrid algorithms like variational quantum eigensolver (VQE) [74, 75]. The gates in VQE are equivalent to tensors with isometric constraints, and might be optimized using our proposed scheme. Besides, this scheme may also be used for stimulating quantum circuits by combining with the recent techniques based on cutting and contracting tensor networks [76, 77].

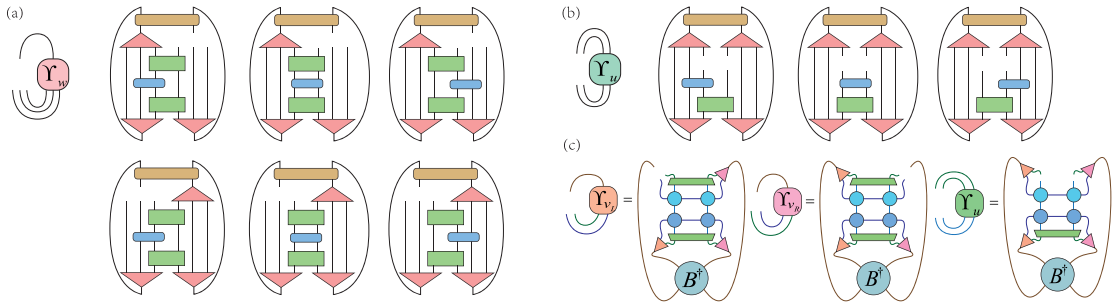

Figure 20. The environment tensors of (a) the isometry w and (b) the disentangler u for MERA, which are the summation of six parts and three parts correspondingly. The environment tensors of v_L, v_R and u in TNR are shown in (c).

Data availability statement

The data that support the findings of this study are openly available at the following URL: https://github.com/xwkgch/IsoTensor.

Acknowledgments

We thank Masaki Oshikawa, Yasuhiro Tada, Atsushi Ueda, Jin-Guo Liu, Song Cheng, Yi-Zhuang You and Markus Hauru for helpful discussions. The simulations were performed using the PyTorch [41] and we have the open source package 'IsoTensor' available on GitHub [78].

C G was supported by JSPS KAKENHI Grant Nos. JP17H06462 and JP19H01808. H Y H is supported by the UC Hellman fellowship. Y Z is supported by the Stanford Q-Farm Bloch Postdoctoral Fellowship in Quantum Science and Engineering.

Appendix A. Evenbly–Vidal optimization method

In order to optimize the whole tensor network, we need to explain how to optimize a single tensor. Here we briefly introduce the Evenbly–Vidal optimization method [31, 35]. Taking the isometry tensor w as example, we need to minimize the energy, or to say loss function

$$E(w) = \mathrm{tr}(wY_w). \tag{A1}$$

We temperately regard w and $w^\dagger$ as independent tensors and Y_w is the environment of the tensor w shown in figure 20(a). By applying the singular value decomposition (SVD) on the environment $Y_w = USV^\dagger$, the energy $E(w)$ can be minimized if we choose the new isometry tensor

$$w' = -VU^\dagger. \tag{A2}$$

Then a single step of updating is to replace the old tensor w by the new tensor w'.

The remaining quest is to determine the environment of the tensor. We can automatically obtain the environment tensors by means of differentiable programming discussed in the main text, or drawing and contracting the environment graphs manually. Here we show the environment graphs for the ternary MERA and TNR used in this paper.

Appendix B. Reducing computation expense of Cayley method

We can reduce the computation of inverting a $n \times n$ matrix to inverting a $2p \times 2p$ ($n > 2p$) matrix by Sherman–Morrison–Woodbury formula

$$\left(B + \alpha UV^T\right)^{-1} = B^{-1} - \alpha B^{-1}U\left(I + \alpha V^T B^{-1}U\right)^{-1}V^T B^{-1}. \tag{B1}$$

Define the concatenated matrices $U = [P_X\bar{X}, X]$ and $V = [X, -P_X\bar{X}]$ where the square brackets here refer to the matrix concatenation and $P_X = I - \frac{1}{2}XX^T$. Then we have $A = UV^T$ and the matrix inverse term in equation (15) becomes

$$\left(I+\frac{\eta}{2}A\right)^{-1}=\left(I+\frac{\eta}{2}UV^T\right)^{-1} \tag{B2}$$

$$=I-\frac{\eta}{2}U\left(I+\frac{\eta}{2}V^TU\right)^{-1}V^T. \tag{B3}$$

And the optimization method of Cayley transform becomes

$$X_{t+1}=X_t-\eta U_t\left(I+\frac{\eta}{2}V_t^TU_t\right)^{-1}V_t^TX_t. \tag{B4}$$

There is an alternative way to accelerate the computation of Cayley transform by the iterative estimation of equation (15) [50, 56]. Note that the fixed point solution of the equation $Y(\eta)=X-\frac{1}{2}\eta A(X+Y(\eta))$ for $Y(\eta)$ is exactly equation (15). By iterating several steps of the equation

$$X_{t+1,0}=X_t-\eta G_X, \tag{B5}$$

$$X_{t+1,k+1}=X_t-\frac{1}{2}\eta A(X_t+X_{t+1,k}), \tag{B6}$$

we can update the tensors in the Cayley transform scheme to avoid computing inverse of matrices.

Appendix C. Training details of MERA

C.1. Resetting mechanism

In the practice of Ising model optimization of MERA, we find that if after the first several hundred iterations the energy error is still larger than a threshold around 10^{-3}, it will be highly possible to be stuck into local minima with energy errors $10^{-3}\sim10^{-4}$. In order to reduce the chance of being stuck into local minima and improve the success rate of optimization, we used a resetting mechanism. To be specific, we first optimize the MERA for 700 iterations and check whether the energy error is larger than a threshold value 1.5×10^{-3}. If the energy error is larger than the threshold value, we reset the optimization to the beginning, meaning that all the trainable parameters in the network are set to there initial values as well as the associated parameters of optimizers.

We note that the Evenbly–Vidal method in our paper has a little difference from Evenbly and Vidal's original algorithm [31]. In the original algorithm, the update is timely, meaning that once a tensor is updated the next derivative or environment tensor is computed using this updated tensor and so on. within the auto-differentiation framework, it is hard to update tensors timely. That is, we compute all the derivative tensors at once. And then we use these derivative tensors to update all the tensors to be optimized.

C.2. Lifting bond dimension trick

In our MERA optimization we use a gradually lifting bond dimension trick which can speed up convergence and improve the accuracy. To be specific, we start the computation with a small bond dimension $\chi=4$ in first 200 iterations. Then we increase the bond dimension to $6,7,8,9,10,12$ sequentially by enlarging the tensors' size and padding zero values. For the result in figure 8, the lifting bond dimension occurs at the $200,700,2700,5700,8700,11700,15700$ iterations.

Appendix D. Methods comparison of MERA for other models

In this section we compare the Evenbly–Vidal method and random mixed method for critical Heisenberg XY model and Heisenberg XXZ model [79]

$$H_{XY}=\sum_r\left(\sigma_x^{[r]}\sigma_x^{[r+1]}+\lambda\sigma_y^{[r]}\sigma_y^{[r+1]}\right), \tag{D1}$$

$$H_{XXZ}=\sum_r\left(\sigma_x^{[r]}\sigma_x^{[r+1]}+\sigma_y^{[r]}\sigma_y^{[r+1]}+\lambda\sigma_z^{[r]}\sigma_z^{[r+1]}\right) \tag{D2}$$

with $\lambda=1.0$.

The results are shown in figure 21. We find that for Heisenberg XY and XXZ models the random mixed method can also improve the optimization performance.

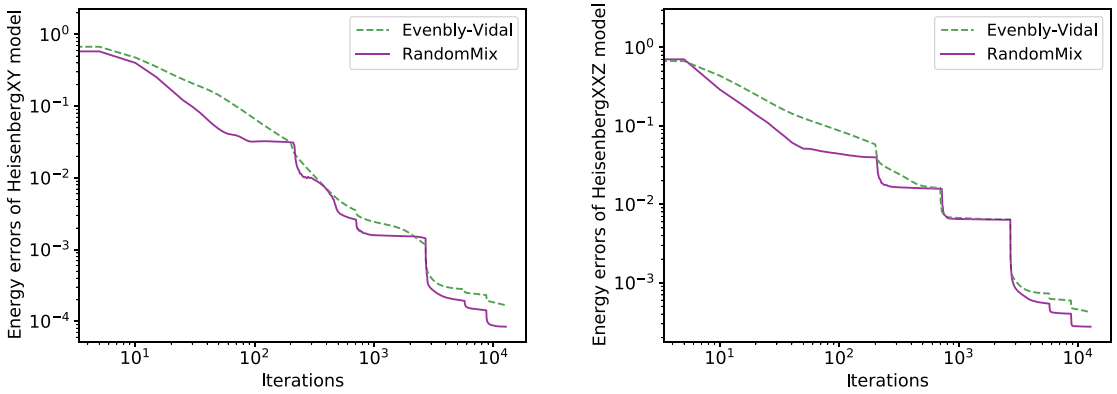

Figure 21. Energy errors with Evenbly–Vidal method and random mixed method for critical Heisenberg XY (left) and XXZ (right) models. The setting of computation is the same as the Ising model in the main text.

Appendix E. Computation graph of TNR

In this section we explicitly describe the computation graph of TNR used in our paper. The computation graph for the TNR is illustrated in figure 16 in the main text.

The computation graph starts from the inverse temperature β because we would like to investigate the partition function with different β. Using equation (33) we can represent the partition function as the contraction of As. In order to prevent the data explosion, the A is renormalized to A_0 by dividing a constant number $A_{\text{norm},0}$ which is taken as the $L2$ norm of A here.

The computation graph of TNR for each layer (gray dashed frame in figure 16) involve three parts. We take the first layer for example:

(a) **Forward propagation**: Taking a 2×2 sub-network of A_0 we make the approximation by inserting projection operators as in figure 11 to obtain the new sub-network $A_{\text{new},0}$. Taking a 2×2 sub-network of A_0 we make the approximation by inserting projection operators as in figure 11 to obtain the new sub-network $A_{\text{new},0}$. Then compare the new sub-network $A_{\text{new},0}$ with the original 2×2 sub-network of A_0 by making contraction of them, we get the approximation error $\|\delta\|$ as the loss function. The forward propagation is shown as gray arrows in figure 16.

(b) **Backward propagation**: From the loss function, the derivative tensors of parameters v_L, v_R and u are automatically computed. With the derivative tensors the parameters can be updated by various methods. Iterating the forward and backward propagation the parameters are optimized. The backward propagation is shown as red arrows in figure 16.

(c) **Tensors renormalization**: Once the parameters have been optimized, we can make the tensors renormalization as described in the main text. The tensors renormalization is shown as brown arrows in figure 16.

By repeating the coarse-graining procedure layer by layer, we finally arrive the top layer and obtain the single tensor A_T in our case. Applying the tensor trace on A_T and collecting A_{norm}s of every layers, we obtain the partition function Z and its logarithm $\ln Z$.

Indeed, the brown arrows in figure 16 also indicate the forward propagation from β to $\ln Z$. We can compute the first-order or even second derivatives of $\ln Z$ with respect to β corresponding to energy density and specific heat in further works.

Appendix F. Soft-constraint optimization

The soft-constraint optimization is to relax the constraints and allow the tensors w and u to be away from isometric, which is known as the method of Lagrange multipliers [48].

Define the elementwise average error

$$T_{\text{err}} = \frac{1}{3}\left(\left\langle I - w^\dagger w\right\rangle + \left\langle I - u^\dagger u\right\rangle + \left\langle I - uu^\dagger\right\rangle\right). \tag{F1}$$

Here the angle bracket refers to the elementwise average of a matrix. The value of T_{err} measures how the the tensors w and u away from the isometric constraints. Adding T_{err} with the energy E we obtain the modified loss function

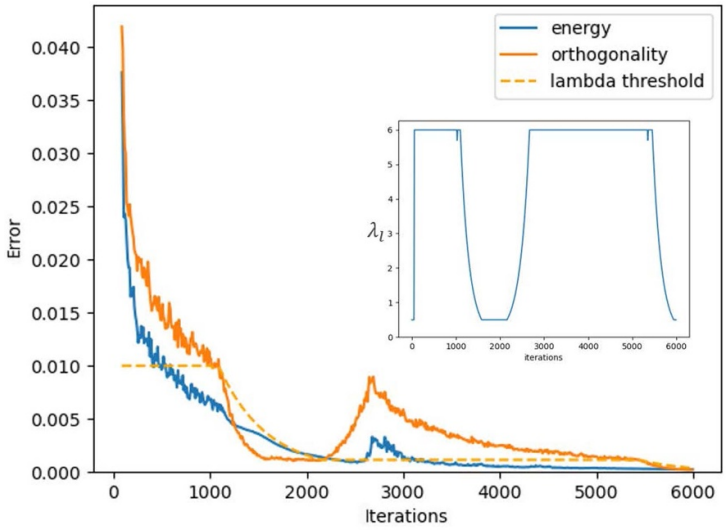

Figure 22. The optimization curves of the soft-constraint method both for the energy error and the orthogonality error. The bond dimension $\chi = 8$. The number of transitional layers is 4.

$$\mathcal{L}_l = E + \lambda_l T_{\text{err}}, \tag{F2}$$

where λ_l is the Lagrange multiplier. Now the constraints are absorbed into the modified loss function and we can optimize the modified loss function $\mathcal{L}_l$ using the usual optimization methods.

The hyperparameter λ_l should be tuned manually and the optimization is sensitive to the hyperparameter λ_l. But the tuning of hyperparameter λ_l is tricky and elusive depending on the systems. We introduce a dynamic tuning process of λ_l here. If λ_l is too large, the optimization will mainly focus on the constraints and the energy decreasing will become difficult, while if λ_l is too small, the optimization will almost ignore the constraints and the energy will keep deceasing. We use a dynamic tuning strategy:

(a) Starting with a small λ_l. After a few iterations switch to a large λ_l.
(b) When the tensors error T_{err} is lower than a threshold, decrease λ_l and the threshold exponentially until to a lower bound.
(c) When the tensors error T_{err} is higher than the threshold, increase λ_l exponentially until to a higher bound.

By this tuning strategy, the hyperparameter λ_l is tuned automatically depending on the tensors error T_{err} such that the optimization keeps a relative balance between the energy E and the tensor error T_{err}.

Figure 22 shows the optimization curves of the soft-constraint method both for the energy error and the orthogonality error by the Adam optimizer. The inner figure shows how the hyperparameter λ_l changes during the optimization process. In this result, the energy error is about $\sim 10^{-5}$, which is larger than what we obtained by Evenbly–Vidal and gradient-based methods in the main text.

ORCID iDs

Chenhua Geng https://orcid.org/0000-0001-8420-0779
Hong-Ye Hu https://orcid.org/0000-0001-5841-831X

References

[1] Hayden P, Nezami S, Qi X-L, Thomas N, Walter M and Yang Z 2016 Holographic duality from random tensor networks *J. High Energy Phys.* **2016** 9
[2] Pastawski F, Yoshida B, Harlow D and Preskill J 2015 Holographic quantum error-correcting codes: toy models for the bulk/boundary correspondence *J. High Energy Phys.* **2015** 149
[3] Verstraete F and Cirac J I 2004 Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions (arXiv:cond-mat/0407066 [cond-mat.str-el])
[4] Vidal G 2008 Class of quantum many-body states that can be efficiently simulated *Phys. Rev. Lett.* **101** 110501
[5] Vidal G 2003 Efficient classical simulation of slightly entangled quantum computations *Phys. Rev. Lett.* **91** 147902
[6] Schollwöck U 2005 The density-matrix renormalization group *Rev. Mod. Phys.* **77** 259
[7] Tilloy A and Cirac J I 2019 Continuous tensor network states for quantum fields *Phys. Rev.* X **9** 021040
[8] Verstraete F, Murg V and Cirac J 2008 Matrix product states, projected entangled pair states and variational renormalization group methods for quantum spin systems *Adv. Phys.* **57** 143

[9] Novikov A, Trofimov M and Oseledets I 2016 Exponential machines (arXiv:1605.03795 [stat.ML])
[10] Stoudenmire E and Schwab D J 2016 Supervised learning with tensor networks *Advances in Neural Information Processing Systems* vol 29, ed D Lee, M Sugiyama, U Luxburg, I Guyon and R Garnett (Curran Associates, Inc.)
[11] Glasser I, Pancotti N and Cirac J I 2018 From probabilistic graphical models to generalized tensor networks for supervised learning (arXiv:1806.05964 [quant-ph])
[12] Martyn J, Vidal G, Roberts C and Leichenauer S 2020 Entanglement and tensor networks for supervised image classification (arXiv:2007.06082 [quant-ph])
[13] Sun Z-Z, Peng C, Liu D, Ran S-J and Su G 2020 Generative tensor network classification model for supervised machine learning *Phys. Rev.* B **101** 075135
[14] Efthymiou S, Hidary J and Leichenauer S 2019 TensorNetwork for machine learning (arXiv:1906.06329 [cs.LG])
[15] Cheng S, Wang L and Zhang P 2020 Supervised learning with projected entangled pair states (arXiv:2009.09932 [cs.CV])
[16] Lu S, Kanász-Nagy M and Cirac J I 2021 Tensor networks and efficient descriptions of classical data (arXiv:2103.06872 [quant-ph])
[17] Cui J, Shi M, Wang H, Yu F, Wu T, Luo X, Ying J and Chen X 2019 Transport properties of thin flakes of the antiferromagnetic topological insulator $MnBi_2Te_4$ *Phys. Rev.* B **99** 155125
[18] Han Z-Y, Wang J, Fan H, Wang L and Zhang P 2018 Unsupervised generative modeling using matrix product states *Phys. Rev.* X **8** 031012
[19] Stokes J and Terilla J 2019 Probabilistic modeling with matrix product states *Entropy* **21** 1236
[20] Glasser I, Sweke R, Pancotti N, Eisert J and Cirac J I 2019 Expressive power of tensor-network factorizations for probabilistic modeling, with applications from hidden Markov models to quantum machine learning (arXiv:1907.03741 [cs.LG])
[21] Gao X, Anschuetz E R, Wang S-T, Cirac J I and Lukin M D 2021 Enhancing generative models via quantum correlations (arXiv:2101.08354 [quant-ph])
[22] Liu J-G and Wang L 2018 Differentiable learning of quantum circuit born machines *Phys. Rev.* A **98** 062324
[23] Kadanoff L P 1966 Scaling laws for ising models near T_c *Phys. Phys. Fiz.* **2** 263
[24] Wilson K G and Kogut J 1974 The renormalization group and the ε expansion *Phys. Rep.* **12** 75
[25] Wilson K G 1983 The renormalization group and critical phenomena *Rev. Mod. Phys.* **55** 583
[26] Hu H-Y, Wu D, You Y-Z, Olshausen B and Chen Y 2020 RG-Flow: a hierarchical and explainable flow model based on renormalization group and sparse prior (arXiv:2010.00029 [cs.LG])
[27] Li S-H and Wang L 2018 Neural network renormalization group *Phys. Rev. Lett.* **121** 260601
[28] Koch-Janusz M and Ringel Z 2018 Mutual information, neural networks and the renormalization group *Nat. Phys.* **14** 578
[29] Li S-H 2021 Learning non-linear wavelet transformation via normalizing flow (arXiv:2101.11306 [cs.LG])
[30] Evenbly G 2019 Number-State preserving tensor networks as classifiers for supervised learning (arXiv:1905.06352 [quant-ph])
[31] Evenbly G and Vidal G 2009 Algorithms for entanglement renormalization *Phys. Rev.* B **79** 144108
[32] Vidal G 2007 Entanglement renormalization *Phys. Rev. Lett.* **99** 220405
[33] Cincio L, Dziarmaga J and Rams M M 2008 Multiscale entanglement renormalization ansatz in two dimensions: quantum Ising model *Phys. Rev. Lett.* **100** 240603
[34] Evenbly G and Vidal G 2015 Tensor network renormalization *Phys. Rev. Lett.* **115** 180405
[35] Evenbly G 2017 Algorithms for tensor network renormalization *Phys. Rev.* B **95** 045117
[36] Levin M and Nave C P 2007 Tensor renormalization group approach to two-dimensional classical lattice models *Phys. Rev. Lett.* **99** 120601
[37] Evenbly G and Vidal G 2015 Tensor network renormalization yields the multiscale entanglement renormalization ansatz *Phys. Rev. Lett.* **115** 200401
[38] Pfeifer R N C, Evenbly G and Vidal G 2009 Entanglement renormalization, scale invariance and quantum criticality *Phys. Rev.* A **79** 040301
[39] Miyaji M, Takayanagi T and Watanabe K 2017 From path integrals to tensor networks for the AdS/CFT correspondence *Phys. Rev.* D **95** 066004
[40] Hu H-Y, Li S-H, Wang L and You Y-Z 2020 Machine learning holographic mapping by neural network renormalization group *Phys. Rev. Res.* **2** 023369
[41] Paszke A *et al* 2019 PyTorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems* vol 32 p 8026
[42] Abadi M *et al* 2016 TensorFlow: a system for large-scale machine learning *12th USENIX Symp. on Operating Systems Design and Implementation (OSDI 16)* (Savannah, GA: USENIX Association) pp 265–83
[43] Liao H-J, Liu J-G, Wang L and Xiang T 2019 Differentiable programming tensor networks *Phys. Rev.* X **9** 031041
[44] Hauru M, Damme M V and Haegeman J 2021 Riemannian optimization of isometric tensor networks *SciPost Phys.* **10** 40
[45] Pascanu R, Mikolov T and Bengio Y 2013 On the difficulty of training recurrent neural networks *Int. Conf. on Machine Learning* (PMLR) pp 1310–8
[46] Polyak B 1964 Some methods of speeding up the convergence of iteration methods *USSR Comput. Math. Math. Phys.* **4** 1–17
[47] Tieleman T *et al* 2012 Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude *COURSERA: Neural Networks for Machine Learning* vol 4 p 26
[48] Bertsekas D P 2014 *Constrained Optimization and Lagrange Multiplier Methods* (New York: Academic)
[49] Tagare H D 2011 Notes on optimization on Stiefel manifolds *Technical Report* (Yale University)
[50] Wen Z and Yin W 2013 A feasible method for optimization with orthogonality constraints *Math. Program.* **142** 397
[51] Absil P-A and Malick J 2012 Projection-like retractions on matrix manifolds *SIAM J. Optim.* **22** 135
[52] Kaneko T, Fiori S and Tanaka T 2013 Empirical arithmetic averaging over the compact Stiefel manifold *IEEE Trans. Signal Process.* **61** 883
[53] Nishimori Y and Akaho S 2005 Learning algorithms utilizing quasi-geodesic flows on the Stiefel manifold *Neurocomputing* **67** 106
[54] Jiang B and Dai Y-H 2015 A framework of constraint preserving update schemes for optimization on Stiefel manifold *Math. Program.* **153** 535
[55] Zhu X 2017 A Riemannian conjugate gradient method for optimization on the Stiefel manifold *Comput. Optim. Appl.* **67** 73
[56] Li J, Fuxin L and Todorovic S 2020 Efficient Riemannian optimization on the Stiefel manifold via the Cayley transform (arXiv:2002.01113 [cs.LG])
[57] Pechen A, Prokhorenko D, Wu R and Rabitz H 2008 Control landscapes for two-level open quantum systems *J. Phys. A: Math. Theor.* **41** 045205

[58] Oza A, Pechen A, Dominy J, Beltrani V, Moore K and Rabitz H 2009 Optimization search effort over the control landscapes for open quantum systems with Kraus-map evolution *J. Phys. A: Math. Theor.* **42** 205305
[59] Manton J 2002 Optimization algorithms exploiting unitary constraints *IEEE Trans. Signal Process.* **50** 635
[60] Evenbly G 2017 Hyperinvariant tensor networks and holography *Phys. Rev. Lett.* **119** 141602
[61] Nozaki M, Ryu S and Takayanagi T 2012 Holographic geometry of entanglement renormalization in quantum field theories *J. High Energy Phys.* **2012** 193
[62] Evenbly G and Vidal G, Quantum criticality with the multi-scale entanglement renormalization ansatz 2013 *Strongly Correlated Systems: Numerical Methods* ed A Avella and F Mancini (Berlin: Springer) pp 99–130
[63] Pfeuty P 1970 The one-dimensional ising model with a transverse field *Ann. Phys., NY* **57** 79
[64] Robbins H and Monro S 1951 A stochastic approximation method *Ann. Math. Stat.* **22** 400
[65] Xie Z Y, Chen J, Qin M P, Zhu J W, Yang L P and Xiang T 2012 Coarse-graining renormalization by higher-order singular value decomposition *Phys. Rev.* B **86** 045139
[66] Yang S, Gu Z-C and Wen X-G 2017 Loop optimization for tensor network renormalization *Phys. Rev. Lett.* **118** 110504
[67] Hauru M, Delcamp C and Mizera S 2018 Renormalization of tensor networks using graph-independent local truncations *Phys. Rev.* B **97** 045111
[68] Onsager L 1944 Crystal statistics. I. A two-dimensional model with an order-disorder transition *Phys. Rev.* **65** 117
[69] Hauru M, Evenbly G, Ho W W, Gaiotto D and Vidal G 2016 Topological conformal defects with tensor networks *Phys. Rev.* B **94** 115125
[70] Hashemizadeh M, Liu M, Miller J and Rabusseau G 2020 Adaptive learning of tensor network structures (arXiv:2008.05437 [cs.LG])
[71] You Y-Z, Yang Z and Qi X-L 2018 Machine learning spatial geometry from entanglement features *Phys. Rev.* B **97** 045153
[72] Hayden P, Nezami S, Qi X-L, Thomas N, Walter M and Yang Z 2016 Holographic duality from random tensor networks *J. High Energy Phys.* **2016** 9
[73] Cong I, Choi S and Lukin M D 2019 Quantum convolutional neural networks *Nat. Phys.* **15** 1273
[74] McClean J R, Romero J, Babbush R and Aspuru-Guzik A 2016 The theory of variational hybrid quantum-classical algorithms *New J. Phys.* **18** 023023
[75] Liu J-G, Zhang Y-H, Wan Y and Wang L 2019 Variational quantum eigensolver with fewer qubits *Phys. Rev. Res.* **1** 023025
[76] Peng T, Harrow A W, Ozols M and Wu X 2020 Simulating large quantum circuits on a small quantum computer *Phys. Rev. Lett.* **125** 150504
[77] Yuan X, Sun J, Liu J, Zhao Q and Zhou Y 2021 Quantum simulation with hybrid tensor networks *Phys. Rev. Lett.* **127** 040501
[78] Geng C 2021 GitHub: differentiable isometric tensor network (available at: https://github.com/xwkgch/IsoTensor)
[79] Fisher M E 1964 Magnetism in one-dimensional systems—the Heisenberg model for infinite spin *Am. J. Phys.* **32** 343