Differentiable Programming Tensor Networks

Hai-Jun Liao,[1,2] Jin-Guo Liu,[1] Lei Wang,[1,2,3,*] and Tao Xiang[1,4,5,†]
[1]Institute of Physics, Chinese Academy of Sciences, Beijing 100190, China
[2]CAS Center for Excellence in Topological Quantum Computation, University of Chinese Academy of Sciences, Beijing 100190, China
[3]Songshan Lake Materials Laboratory, Dongguan, Guangdong 523808, China
[4]University of Chinese Academy of Sciences, Beijing 100049, China
[5]Collaborative Innovation Center of Quantum Matter, Beijing 100190, China

(Received 2 April 2019; published 5 September 2019)

Differentiable programming is a fresh programming paradigm which composes parameterized algorithmic components and optimizes them using gradient search. The concept emerges from deep learning but is not limited to training neural networks. We present the theory and practice of programming tensor network algorithms in a fully differentiable way. By formulating the tensor network algorithm as a computation graph, one can compute higher-order derivatives of the program accurately and efficiently using automatic differentiation. We present essential techniques to differentiate through the tensor networks contraction algorithms, including numerical stable differentiation for tensor decompositions and efficient backpropagation through fixed-point iterations. As a demonstration, we compute the specific heat of the Ising model directly by taking the second-order derivative of the free energy obtained in the tensor renormalization group calculation. Next, we perform gradient-based variational optimization of infinite projected entangled pair states for the quantum antiferromagnetic Heisenberg model and obtain state-of-the-art variational energy and magnetization with moderate efforts. Differentiable programming removes laborious human efforts in deriving and implementing analytical gradients for tensor network programs, which opens the door to more innovations in tensor network algorithms and applications.

DOI: 10.1103/PhysRevX.9.031041

Subject Areas: Computational Physics, Condensed Matter Physics

I. INTRODUCTION

Tensor networks are prominent approaches for studying classical statistical physics and quantum many-body physics problems [1–3]. In recent years, its application has expanded rapidly to diverse regions, including the simulation and design of quantum circuits [4–7], quantum error correction [8,9], machine learning [10–14], language modeling [15,16], quantum field theory [17–20], and holographic duality [21,22].

One of the central problems relevant to many research directions is the optimization of tensor networks in a general setting. Despite highly successful optimization schemes for one-dimensional matrix product states [23–28], optimizing tensor networks in two or higher dimensions has been a challenging topic. This difficulty is partly due to the high computational cost of tensor contractions and partly due to the lack of an efficient optimization scheme in the high-dimensional situation.

The challenge is particularly pressing in optimizing tensor network states for infinite translational-invariant quantum systems. In these cases, the same tensor affects the variational energy in multiple ways, which results in a highly nonlinear optimization problem. Optimization schemes based on approximate imaginary time projection [29–33] struggle to address nonlocal dependence in the objective function. References [34,35] applied gradient-based optimization and showed that it significantly improved the results. However, it is cumbersome and error prone to derive the gradients of tensor network states analytically, which involves multiple infinite series of tensors even for a simple physical Hamiltonian. This fact has limited broad application of gradient-based optimization of tensor network states to more complex systems. Alternative approaches, such as computing the gradient using numerical derivatives, have limited accuracy and efficiency; therefore, they only apply to cases with few variational parameters [36,37]. Deriving the gradient manually using the chain rule is only manageable for purposely designed simple tensor network structures [38].

*wanglei@iphy.ac.cn
†txiang@iphy.ac.cn

Differentiable programming provides an elegant and efficient solution to these problems by composing the whole tensor network program in a fully differentiable manner and optimizing them using automatic differentiation.In this paper, we present essential automatic differentiation techniques that compute (higher-order) derivatives of tensor network programs efficiently to numeric precision. This progress opens the door to gradient-based (and even Hessian-based) optimization of tensor network states in a general setting. Moreover, computing (higher-order) gradients of the output of a tensor network algorithm offers a straightforward approach to compute physical quantities such as the specific heat and magnetic susceptibilities. The differentiable programming approach is agnostic to the detailed lattice geometry, Hamiltonian, and tensor network contraction schemes. Therefore, the approach is general enough to support a wide class of tensor network applications.

We focus on applications that involve two-dimensional infinite tensor networks where the differentiable programming techniques offer significant benefits compared to the conventional approaches. We show that after solving major technical challenges such as numerical stable differentiation through singular value decomposition (SVD) and memory-efficient implementation for fixed-point iterations, one can obtain the state-of-the-art results in variational optimization of tensor network states.

The organization of this paper is as follows. In Sec. II, we introduce automatic differentiation in the context of tensor network algorithms and formulate tensor network contractions as computation graphs. In Sec. III, we present the key techniques for stable and scalable differentiable programming of tensor network algorithms. In Sec. IV, we demonstrate the ability of the approach with applications to the classical Ising model and the quantum Heisenberg model on the infinite square lattice. Finally, we give our outlook for future research directions in Sec. V. Our code implementation is publicly available (see Ref. [39]).

II. GENERAL THEORY

Automatic differentiation through a computation graph is a unified framework that covers training neural networks for machine learning, optimizing tensor networks for quantum physics, and many more. We first review the core idea of automatic differentiation and then explain its application to various tensor network contraction algorithms formulated in terms of computation graphs.

A. Automatic differentiation

Automatic differentiation mechanically computes derivatives of computation processes expressed in terms of computer programs [40]. Unlike numerical differentiation, automatic differentiation computes the value of derivatives to machine precision. Its performance has a general theoretical guarantee, which does not exceed the algorithmic complexity of the original program [41,42].

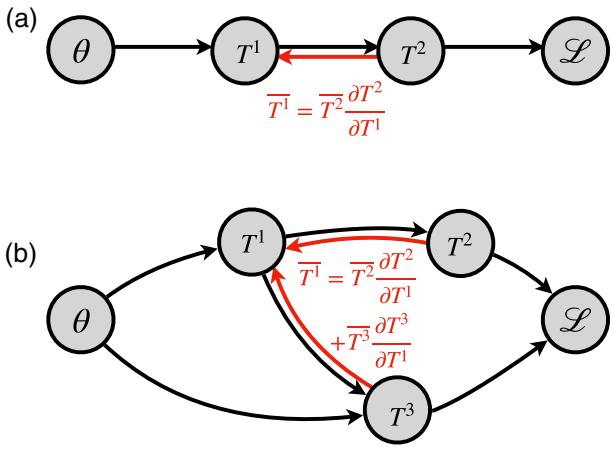


FIG. 1. Reverse-mode automatic differentiation on computation graphs. Black arrows indicate the forward function evaluation from inputs to outputs. Red arrows indicate backward steps for adjoint backpropagation. (a) A chain graph. (b) A more general computation graph. In the backward pass, the adjoint of a given node is computed according to Eq. (2).

Automatic differentiation is the computational engine of modern deep learning applications [43,44]. Moreover, it also finds applications in quantum optimal control [45] and quantum chemistry calculations such as computing forces [46] and optimizing basis parameters [47].

Central to the automatic differentiation is the concept of the computation graph. A computation graph is a directed acyclic graph composed of elementary computation steps. The nodes of the graph represent data, which can be scalars, vectors, matrices, or tensors [48]. The graph connectivity indicates the dependence of the data flow in the computation process. The simplest computation graph is the chain shown in Fig. 1(a). Starting from, say, vector-valued input parameters θ, one can compute a series of intermediate results until reaching the final output $\mathcal{L}$, which we assume to be a scalar. The so-called forward evaluation simply traverses the chain graph in sequential order, $\theta \to T^1 \to \cdots \to T^n \to \mathcal{L}$.

To compute the gradient of the objective function with respect to input parameters, one can exploit the chain rule

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial T^n}\frac{\partial T^n}{\partial T^{n-1}}\cdots\frac{\partial T^2}{\partial T^1}\frac{\partial T^1}{\partial \theta}. \tag{1}$$

Since we consider the case where the input dimension is larger than the output dimension, it is more efficient to evaluate the gradient in Eq. (1) by multiplying terms from left to right using a series of vector-Jacobian products. In terms of the computation graph shown in Fig. 1(a), one traverses the graph backward and propagates the gradient signal from the output back to the input. Computing the derivative this way is called reverse-mode automatic differentiation. This approach, commonly referred to as the

backpropagation algorithm [43], is arguably the most successful method for training deep neural networks.

It is instructive to introduce the adjoint variable $\bar{T} = \partial\mathcal{L}/\partial T$ to denote the gradient of the final output $\mathcal{L}$ with respect to the variable T. One sees that the reverse-mode automatic differentiation propagates the adjoint from $\overline{T^n} = \bar{\mathcal{L}}\partial\mathcal{L}/\partial T^n$, with $\bar{\mathcal{L}} = 1$, all the way back to $\overline{T^i} = \overline{T^{i+1}}\partial T^{i+1}/\partial T^i$, with $i = n-1, \ldots, 1$, and finally computes $\bar{\theta} = \overline{T^1}\partial T^1/\partial\theta$. In each step, one propagates the adjoint backward via a local vector-Jacobian product.

The adjoint backpropagation picture generalizes well to more complex computation graphs. For example, the data node T^1 in Fig. 1(b) affects the final output via two different downstream computation paths. In the backward pass, one needs to accumulate all contributions from its child nodes for its adjoint. In general, the backpropagation rule reads

$$\overline{T^i} = \sum_{j:\ \text{child of } i} \overline{T^j}\frac{\partial T^j}{\partial T^i}. \tag{2}$$

The reverse-mode automatic differentiation algorithm can be understood as a message-passing process on the computation graph. After a topological sort of the computation graph defined by the forward pass, one visits the graph backward from the output node with adjoint $\bar{\mathcal{L}} = 1$. Each node collects information from its child nodes to compute its own adjoint and then passes this information to its parents. Thus, one can compute the gradient with respect to all parameters in one forward and one backward pass. Typically, one caches necessary information in the forward pass for efficient evaluation of the vector-Jacobian product in the backward pass.

The building blocks of a differentiable program are called primitives. The primitives can be elementary operations such as addition, multiplication, and math functions [50]. Each primitive has an associated backward function in addition to the ordinary forward function. The backward function backpropagates the adjoints according to the vector-Jacobian product Eq. (2). Note that one does not need to explicitly instantiate or store the full Jacobian matrices. Moreover, one can group many elementary computation steps together as a primitive. For example, the linear algebra operations such as matrix multiplication can be regarded as a primitive. In this way, the forward pass of these customized primitives can be operated as a black box. Designing the differentiable program in such a modular way allows one to control the level of granularity of automatic differentiation. There are several advantages of crafting customized primitives for domain-specific problems. First, this can reduce the computation and memory cost. Second, in some cases, it is numerically more stable to group several steps together. Third, one can wrap function calls to external libraries into primitives, without the need to track each individual computation step.

Modern machine learning frameworks support automatic differentiation via various mechanisms. For example, TensorFlow [51] explicitly constructs computation graphs using a customized language; autograd [52], PyTorch [53], and Jax [54] track the program execution order at run time; and Zygote [55] performs source code transformation at the compiler level. These frameworks allow one to differentiate through function calls, control flows, loops, and many other computation instructions. Building on these infrastructures, differentiable programming is emerging as a new programming paradigm that emphasizes assembling differentiable components and learning the whole program via end-to-end optimization [44]. Letting machines deal with automatic differentiation mechanically has greatly reduced laborious human efforts and reallocated human attention to design more sophisticated and creative deep learning algorithms.

Finally, we note that it is also possible to evaluate Eq. (1) from right to left, which corresponds to the forward-mode automatic differentiation. The operation of the forward-mode automatic differentiation is akin to perturbation theory. One can compute the objective function and the gradient in a single forward pass without storing any intermediate results. However, the forward-mode automatic differentiation is not favorable for computation graphs whose input dimension is much larger than the output dimension [44]. Therefore, the majority of deep learning work employs the reverse-mode automatic differentiation. For the same reason, we focus on the reverse-mode automatic differentiation for tensor network programs.

B. Computation graphs of tensor network contractions

A tensor network program maps input tensors to an output, which we assume to be a scalar. Depending on the context, the input tenors may represent classical partition functions or quantum wavefunctions, and the outputs can be various physical quantities of interest. It is conceptually straightforward to apply automatic differentiation to a tensor network program by expressing the computation and, in particular, the tensor network contractions as a computation graph.

As a pedagogic example, consider the partition function of the infinite one-dimensional Ising model $Z = \lim_{N\to\infty}\mathrm{Tr}(T^N)$, where $T = \begin{pmatrix} e^{\beta} & e^{-\beta} \\ e^{-\beta} & e^{\beta} \end{pmatrix}$ is a second-rank tensor (matrix) representing the Boltzmann weight. One can numerically access the partition function in the thermodynamic limit by repeatedly squaring the matrix T and tracing the final results. The computation graph has a simple chain structure shown in Fig. 1(a). Differentiating with respect to such a computational process involves backpropagating the adjoint through matrix trace and multiplication operations, which is straightforward.

At this point, it is also worth distinguishing between the exact and approximate tensor network contraction schemes. Tensor networks on a treelike graph can be contracted

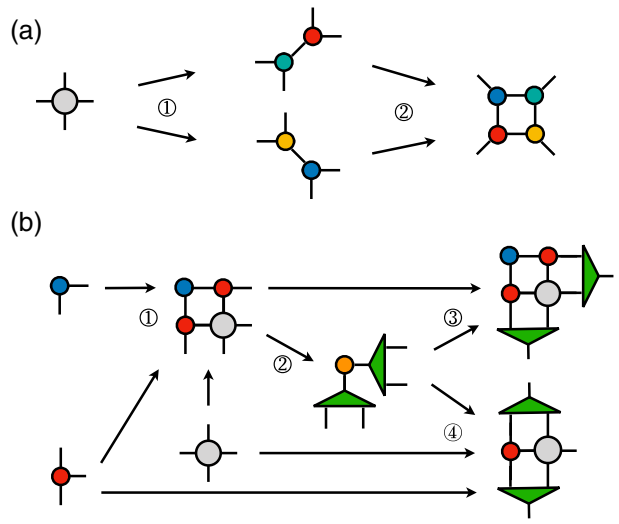


FIG. 2. (a) The iteration step of TRG. (b) The iteration step of CTMRG. Each tensor is a node in the computation graph. The primitive functions in the computation graphs are SVD and tensor contractions.

exactly and efficiently. However, other exact approaches, such as the one used for counting and simulating quantum computing [56,57], in general, exhibit exponentially scaling with the problem size. Nevertheless, it is straightforward to apply automatic differentiation to these exact algorithms since they mostly involve tensor reshape and contractions.

We focus on the less trivial cases of differentiating through *approximate* tensor contractions, which typically involve truncated tensor factorization or variational approximations. They cover important tensor network applications, which show great advantages over other numerical methods [1–3]. In particular, we are interested in contracting infinite tensor networks, where the fundamental data structure is the bulk tensor. The contraction schemes loosely fall into three categories: those based on coarse-graining transformations [58–64], those based on the corner transfer matrix [65–67], and those based on matrix product states [2,24,68]. Since the last two contraction schemes are closely related [2], in the following, we focus on automatic differentiation for the tensor renormalization group (Sec. II B 1) and corner transfer matrix renormalization group approaches (Sec. II B 2), respectively.

1. Tensor renormalization group

The tensor renormalization group (TRG) method contracts the tensor network by factorizing and blocking the bulk tensors iteratively [58]. Figure 2(a) shows one step of the TRG iteration as the computation graph, which includes (1) splitting the bulk tensor in two ways using SVD, where we have truncated the singular values and vectors to a prescribed bond dimension χ, and (2) assembling the four 3-leg tensors generated in the last step into a 4-leg tensor. After this contraction, we obtain a new tensor for the next iteration.

The TRG method grows the lattice size exponentially fast. So, one quickly reaches the thermodynamic limit after a few tens of iterations. Note that for numerical stability, one needs to rescale the tensor elements after each iteration. The computational cost of the TRG method scales as $\mathcal{O}(\chi^6)$, and the memory cost scales as $\mathcal{O}(\chi^4)$. After unrolling the iterations, the computation graph of the TRG method is similar to the simple chain graph shown in Fig. 1(a). Within each iteration step, the basic operations are tensor index permutation, truncated SVD, and tensor contractions. Since each of these operations is differentiable, one can backpropagate through the TRG procedure to compute the derivative of a downstream objective function with respect to the input tensor.

2. Corner transfer matrix renormalization group

The computation graph of the corner transfer matrix renormalization group (CTMRG) [65] has a more interesting topology. The goal of CTMRG calculation is to obtain converged corner and edge tensors that represent the environment degrees of freedom (d.o.f.) of the bulk tensor.

In cases where the bulk tensor has the full symmetry of the square lattice, the steps of one CTMRG iteration is shown in Fig. 2(b): (1) Contract the bulk tensor with the corner and edge tensors to form a 4-leg tensor, (2) perform truncated SVD to the 4-leg tensor, keeping the singular dimensions up to the cutoff χ (keep the truncated singular matrix as the isometric projector); (3) apply the isometry to the 4-leg tensor from the first step to find a new corner tensor; (4) apply the same isometry to find a new edge tensor for the next step. Iterate this procedure until convergence. One sees that the same bulk tensor with bond dimension d appears in each step of the CTMRG iteration. For this reason, the converged environment tensors will depend on the bulk tensor in a complicated way.

Unlike the TRG method [58], the CTMRG approach grows the system size linearly. So, one may need to iterate a bit more steps to reach convergences in CTMRG. On the other hand, the computational complexity $\mathcal{O}(d^3\chi^3)$ and memory cost $\mathcal{O}(d^2\chi^2)$ of CTMRG are smaller than the ones of TRG in terms of the cutoff bond dimension.

III. TECHNICAL INGREDIENTS

To compute gradients of a tensor network program using reverse-mode automatic differentiation, one needs to trace the composition of the primitive functions and propagate the adjoint information backward on the computation graph. Thankfully, modern differentiable programming frameworks [51–55] have taken care of tracing and backpropagation for their basic data structure, differentiable tensors, automatically.

What one needs to focus on is to identify suitable primitives of tensor network programs and define their vector-Jacobian products for backpropagation. The key

components of tensor network algorithms are the matrix and tensor algebras. There are established results on backpropagation through these operations [69–71]. First of all, it is straightforward to wrap all BLAS routines as primitives with customized backward functions. Next, although less trivial, it is also possible to derive backward rules for many LAPACK routines such as the eigensolver, SVD, and QR factorization [69]. By treating these linear algebra operations as primitives, one can compose a differentiable program with efficient implementation of matrix libraries.

However, there are a few practical obstacles to stable and scalable implementation of differentiable tensor network programs. First, the backward pass for the eigensolver and SVD may face numerical instability with degeneracy in the eigenvalues or singular values. Second, the reverse-mode automatic differentiation may incur large memory consumption, which prevents one from reaching the same bond dimension of an ordinary tensor network program. We present solutions to these problems below.

A. Stable backward pass through linear algebra operations

We present several key results on matrix derivatives involving linear algebra operations that are relevant to tensor network algorithms. Thanks to the modular nature of reverse-mode automatic differentiation, one just needs to specify the local backward function to integrate these components into a differentiable program. We comment on their connections to the physics literature and pay special attention to stable numerical implementations [39]. For more information, one can refer to Refs. [69–71].

1. Symmetric eigensolver

The forward pass reads $A = U\Lambda U^T$, where Λ is the diagonal matrix of eigenvalues λ_i and each column of the orthogonal matrix U is a corresponding eigenvector. In the computation graph, the node A has two child nodes U and Λ.

In the backward pass, given the adjoint $\bar{U}$ and $\bar{\Lambda}$, we have [69,71]

$$\bar{A} = U[\bar{\Lambda} + F \odot (U^T\bar{U} - \bar{U}^T U)/2]U^T, \tag{3}$$

where $F_{ij} = (\lambda_j - \lambda_i)^{-1}$ if $i \neq j$ and zero otherwise. The symbol $\odot$ denotes an elementwise Hadamard product. One can readily check that the gradient is also a symmetric matrix.

Equation (3) can be regarded as "reverse" perturbation theory. When the downstream calculation does not depend on the eigenstate, i.e., $\bar{U} = 0$, the backward equation is related to the celebrated Hellmann-Feynman theorem [72], which connects the perturbation to the Hamiltonian and its eigenvalues. Reference [73] applied the special case of Eq. (3) to an inverse Hamiltonian design based on energy spectra.

The appearance of the eigenvalue difference in the denominator of F is a reminder of the first-order nondegenerate perturbation theory. Reference [47] was concerned about the stability of the backpropagation through the eigensolver, thus turned to less efficient forward-mode automatic differentiation for variational optimization of the Hartree-Fock basis. In many physical problems, the final objective function depends on the eigenvalues and eigenstates in a gauge-independent way, e.g., a quadratic form of occupied eigenstates. In these cases, only the eigenvalue difference between the occupied and unoccupied states will appear in the denominator of F, which is a familiar pattern in the linear response theory [74]. Therefore, degenerate eigenvalues would not necessarily cause a problem for these physical applications [75]. In practice, we found that, by using a Lorentzian broadening with $1/x \to x/(x^2+\varepsilon)$ with $\varepsilon = 10^{-12}$, one can stabilize the calculation at the cost of introducing a small error in the gradient; see also Ref. [71].

2. Singular value decomposition

A ubiquitous operation in tensor network algorithms is the matrix SVD, which is used for canonicalization and factorization of tensor networks [1,2,25]. The forward pass reads $A = USV^T$, where A is of the size (m, n), and U, V^T has size (m, k) and (k, n), respectively, and $k = \min(m, n)$. Note that S is a diagonal matrix that contains singular values s_i. In the reverse-mode automatic differentiation, given the adjoints $\bar{U}, \bar{S}$ and $\bar{V}$, one can obtain [70]

$$\begin{aligned}\bar{A} = &\frac{1}{2}U[F_+ \odot (U^T\bar{U} - \bar{U}^T U) + F_- \odot (V^T\bar{V} - \bar{V}^T V)]V^T \\ &+ U\bar{S}V^T + (I - UU^T)\bar{U}S^{-1}V^T + US^{-1}\bar{V}^T(I - VV^T),\end{aligned} \tag{4}$$

where $[F_\pm]_{ij} = \{[1/(s_j - s_i)] \pm [1/(s_j + s_i)]\}$ for $i \neq j$ and zero otherwise. To prevent the numerical issue in the case of degenerate singular values, we use the same Lorentzian broadening as Sec. III A 1 for the first term, which works well in our experience. In practice, for variational tensor network calculation starting from random tensors, the chance of having exact degenerate eigenvalues is small. Even if this happens, applying the rounding is a reasonable solution. However, for the case of degeneracy due to intrinsic reasons [60,62,76,77], one will still obtain the correct gradient as long as the end-to-end gradient is well defined. Lastly, inverting the vanishing singular values in Eq. (4) is not a concern since the corresponding space is usually truncated.

3. QR factorization

QR factorization is often used for canonicalization of tensor networks [1,2,25]. In the forward pass, one

factorizes $A = QR$, where $Q^TQ = I$ and R is an upper triangular matrix [78]. Depending on the dimensions (m, n) of the matrix A there are two cases for the backward function.

For the input shape of the A matrix $m \geq n$, R is an $n \times n$ matrix. The backward pass reads [71]

$$\bar{A} = [\bar{Q} + Q\texttt{copyltu}(M)]R^{-T}, \tag{5}$$

where $M = R\bar{R}^T - \bar{Q}^TQ$ and the copyltu function generates a symmetric matrix by copying the lower triangle of the input matrix to its upper triangle, $[\texttt{copyltu}(M)]_{ij} = M_{\max(i,j),\min(i,j)}$. We can handle the multiplication of R^{-T} by solving a linear system with a triangular coefficient matrix.

For the case of $m < n$, Q is of the size (m, m), and R is an $m \times n$ matrix. We denote $A = (X, Y)$ and $R = (U, V)$, where X and U are full-rank square matrices of size (m, m). This decomposition can be separated into two steps: First, $X = QU$ uniquely determines Q and U, and then we calculate $V = Q^TY$. Applying the chain rule, the backward rule gives

$$\bar{A} = ([(\bar{Q} + \bar{V}Y^T) + Q\texttt{copyltu}(M)]U^{-T}, Q\bar{V}), \tag{6}$$

where $M = U\bar{U}^T - (\bar{Q} + \bar{V}Y^T)^TQ$.

B. Memory-efficient reverse-mode automatic differentiation with checkpointing function

A straightforward implementation of the reverse-mode automatic differentiation for tensor networks has a large memory overhead because one needs to store intermediate results in the forward pass for evaluating the vector-Jacobian products in the backward pass. The number of stored variables is related to the level of granularity in the implementation of the automatic differentiation. In any case, the memory consumption of reverse-mode automatic differentiation will be proportional to the depth of the computation graph. This overhead is particularly worrisome for tensor networks with large bond dimensions and a large number of renormalization group (RG) iterations.

The solution to the memory issue of reverse-mode automatic differentiation is a well-known technique called checkpointing [50,79]. The idea is to trade the computational time for memory usage. Taking the chain computation graph in Eq. (1) as an example, one can store the tensor every few steps in the forward process. In the backward pass, one recomputes intermediate tensors whenever needed by running a small segment of the computation graph forward. In this way, one can greatly reduce the memory usage with no more than twice the computational effort.

In another perspective, checkpointing amounts to defining customized primitives, which encapsulates a large part of the computation graph. These primitives have their own special backward rules that locally run the forward pass again and then backpropagate the adjoint. Therefore, in the forward pass, one does not need to cache internal states of these checkpointing primitives.

Checkpointing is a general strategy that is applied to the computation graph of any topological structure. When applied to tensor network algorithms, it is natural to regard the renormalization steps shown in Fig. 2 as checkpointing primitives. In this way, one avoids storing some large intermediate tensors in the forward pass.

C. Backward through fixed-point iteration

Fixed-point iteration is a recurring pattern in tensor network algorithms. For example, one iterates the function $T^{i+1} = f(T^i, \theta)$ until reaching a converged tensor T^* and uses it for downstream calculations. To compute the gradient with respect to the parameter θ, one can certainly unroll the iteration to a deep computation graph and directly apply the reverse-mode automatic differentiation. However, this approach has the drawback of consuming large memory if it takes long iterations to find the fixed point.

One can solve this problem by using the implicit function theorem on the fixed-point equation [80]. Taking the derivative on both sides of $T^* = f(T^*, \theta)$, we have

$$\begin{aligned}\bar{\theta} = \overline{T^*}\frac{\partial T^*}{\partial \theta} &= \overline{T^*}\left[I - \frac{\partial f(T^*,\theta)}{\partial T^*}\right]^{-1}\frac{\partial f(T^*,\theta)}{\partial \theta} \\ &= \sum_{n=0}^{\infty}\overline{T^*}\left[\frac{\partial f(T^*,\theta)}{\partial T^*}\right]^n\frac{\partial f(T^*,\theta)}{\partial \theta}.\end{aligned} \tag{7}$$

The second line expands the matrix inversion in the square brackets as a geometric series. Therefore, to backpropagate through a fixed-point iteration, the basic operation involves just the vector-Jacobian products with the single-step iteration function. In the backward function, one performs iteration to accumulate the adjoint $\bar{\theta}$ until reaching its convergence. The geometric series shows the same convergence rate to the fixed point as the forward iteration [80].

Many of the tensor network contraction schemes, including the CTMRG method reviewed in Sec. II B 2, fall into the framework of fixed-point iterations. Thus, one can use Eq. (7) for backward through CTMRG calculations, where the iteration is over the RG step shown in Fig. 2(b). We note that the analytical gradient of infinite tensor network contraction derived in Refs. [34,35] contains a similar pattern, which is a summation of geometric series.

Similar to the checkpoint technique of Sec. III B, Eq. (7) also reduces the memory usage in the reverse-mode automatic differentiation since one does not need to store a long chain of intermediate results in the forward iteration. Moreover, since the downstream objective function is independent of how the fixed-point tensor is obtained, one can also exploit an accelerated iteration scheme [81] in

the forward process [82]. However, there is a caveat when applying Eq. (7) to differentiating tensor network RG algorithms. One may need to pay special attention to the redundant global gauge in the RG iterations to ensure that the fixed-point equation indeed holds.

D. Higher-order derivatives

Since the gradient can also be expressed as a computation graph, one can compute the second-order derivatives by applying automatic differentiation to the graph again. In this way, one can, in principle, compute arbitrary higher-order derivatives of a program using automatic differentiation [50]. Deep learning frameworks [51–55] have out-of-the-box support for computing higher-order derivatives.

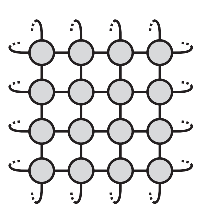
The ability to compute higher derivatives supports Hessian-based optimization of tensor network states, such as the Newton method [83]. However, computing and inverting the full Hessian matrix explicitly could be prohibitively expensive and unnecessary. One can efficiently compute the Hessian-vector product via $\sum_j[(\partial^2\mathcal{L})/(\partial\theta_i\partial\theta_j)]x_j = [\partial/(\partial\theta_i)]\{\sum_j(\partial\mathcal{L})/(\partial\theta_j)]x_j\}$ without constructing the Hessian matrix explicitly [84]. This process is sufficient for iterative linear equation solvers used for the Newton method.

IV. APPLICATIONS

We present two applications to demonstrate the versatility of the differentiable programming tensor network approach for statistical physics and quantum many-body problems. Our publicly available code implementation [39] employs PyTorch [85] with a customized linear algebra automatic differentiation library for improved numerical stability (see discussions in Sec. III A). However, we note that one can readily reproduce the results with other modern deep learning frameworks such as autograd [86], TensorFlow [87], Jax [88], and Zygote [89].

A. Higher-order derivative of the free energy

Consider an Ising model on the square lattice with inverse temperature β; its partition function can be expressed as a two-dimensional tensor network with bond dimension $d = 2$,

$Z =$  . (8)

The bulk tensor is [90]

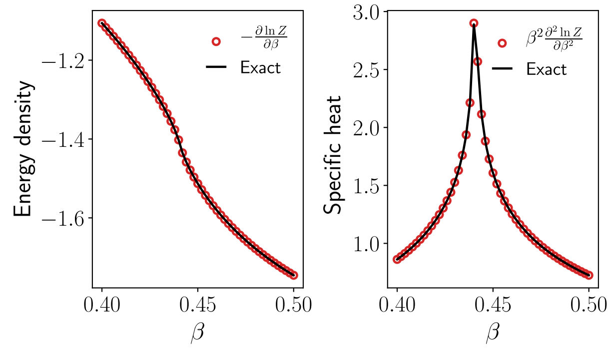
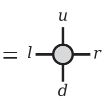


FIG. 3. Energy density and specific heat of the 2D Ising model. They are computed by taking the first- and second-order derivatives of the free energy obtained after 30 TRG iteration steps with a cutoff bond dimension $\chi = 30$. Solid lines are exact solutions [93].

$T_{uldr} =$  $= \frac{\sqrt{\lambda_u\lambda_l\lambda_d\lambda_r}}{2}\,\delta_{\mathrm{mod}(u+l-d-r,2)}$, (9)

where $\lambda_u = e^{\beta} + (-1)^u e^{-\beta}$. We contract the infinite tensor network using the TRG approach discussed in Sec. II B 1. We use a cutoff bond dimension $\chi = 30$ and iterate for 30 TRG steps. Finally, we obtain the partition function Eq. (8) and the free energy by tracing out the bulk tensor.

Next, we compute the physical observables such as energy density and specific heat by directly taking derivatives of the free energy using automatic differentiation, as shown in Fig. 3. Note that the energy density shows a kink, and the specific heat exhibits a peak around the critical temperature $\beta_c = \ln(1+\sqrt{2})/2 \approx 0.44068679$. Unlike numerical differentiation, these results are free from the finite difference error [61,91]. Accurate computation of higher-order derivatives of the tensor network algorithm will be useful to investigate thermal and quantum phase transitions. We note that it is, in principle, possible to obtain the specific heat by directly computing the energy variance [35,92], which, however, involves cumbersome summation of geometric series expressed in terms of tensor networks.

There are alternative ways to compute the specific heat with automatic differentiation. For example, one can directly compute the energy by using the impurity tensor and then take the first-order derivative to obtain the specific heat. Alternatively, one can also use forward-mode automatic differentiation since there is only one input parameter β to be differentiated. We have purposely chosen the present approach to highlight the power of differentiable programming with the reverse-mode automatic differentiation technique. Backpropagating through the whole TRG procedure, and, in particular, the SVD, allows one to compute physical observables using higher-order derivatives. It is remarkable that this method works, given many of the degenerate singular values due to the Z_2 symmetry of the Ising model [47]. To obtain correct physical results, it is

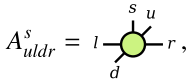
crucial to implement the SVD backward function in a numerically stable way as explained in Sec. III A 2.

B. Gradient-based optimization of iPEPS

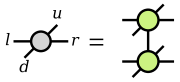
We consider a variational study of the square lattice antiferromagnetic Heisenberg model with the Hamiltonian

$$H = \sum_{\langle i,j \rangle} S_i^x S_j^x + S_i^y S_j^y + S_i^z S_j^z. \tag{10}$$

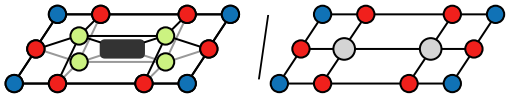
We consider an infinite projected entangled pair state (iPEPS) as the variational ansatz. The variational parameters are the elements in the iPEPS,

$$A^s_{uldr} =$$  , (11)

where s denotes the physical indices, and the remaining indices u, l, d, r are for virtual d.o.f. of the bond dimension D. We initialize the tensor elements with random Gaussian variables. The overlap of the iPEPS forms a tensor network, where the bulk tensor is the double-layer tensor with squared bond dimension D^2,

$$T_{uldr} =$$  . (12)

To contract the infinite tensor network formed by this bulk tensor, we use the CTMRG method reviewed in Sec. II B 2. We initialize the corner and edge tensors by partially tracing out legs from the bulk tensor; we then perform the CTMRG iteration until we reach convergence in the corner and edge tensors. After contraction, we can evaluate the expected energy $\langle\psi|H|\psi\rangle/\langle\psi|\psi\rangle$. Because of the translational invariance of the problem, it is sufficient to consider the expected energy on a bond,

$$\mathcal{L} =$$  , (13)

where the black rectangle in Eq. (13) is the Hamiltonian operator acting on a bond. We have performed a basis rotation to the Hamiltonian so that the ground state will have a single-site unit cell. We use cutoff bond dimensions $\chi = 30, 50, 80, 100, 144, 160$ for $D = 2, 3, \ldots, 7$, respectively. Since the expected energy decreases with the cutoff dimension [36,94], the approximated CTMRG contraction gives a variational upper bound to the ground-state energy. The expected energy in Eq. (13) has both explicit and implicit dependence on the variational parameters in Eq. (11) via the corner and edge tensors.

We compute the gradient of Eq. (13) with respect to the single-layer tensor Eq. (11) using automatic differentiation, which automatically resolves the intricate dependence

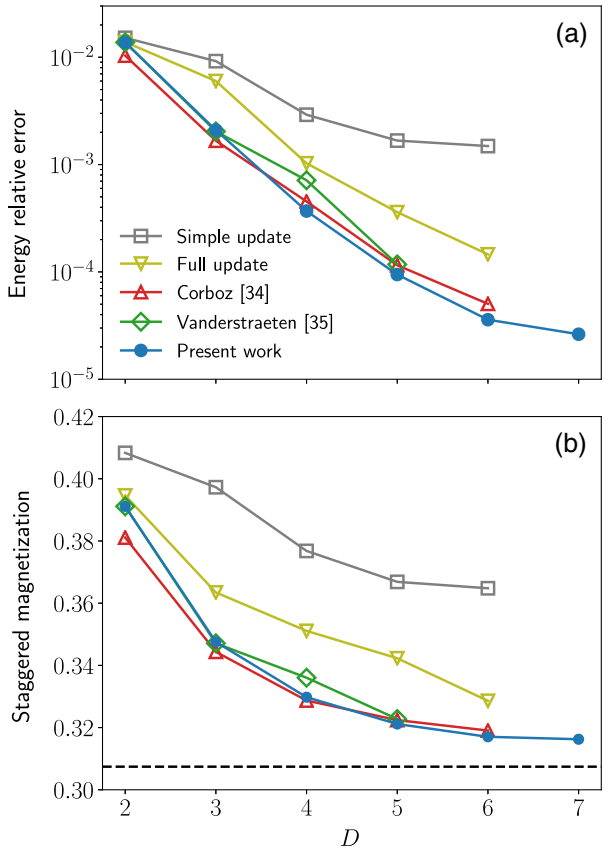


FIG. 4. (a) The relative error in the energy of the 2D $S = 1/2$ antiferromagnetic Heisenberg model compared to previous variational results [34,35]. The accuracy is measured relative to the extrapolated QMC result [95]. (b) A comparison of the staggered magnetization, where the dashed line is the extrapolated QMC result [95]. The simple and full update reference data are also from Ref. [34].

structure in the computation graph of Fig. 2(b). The gradient computation takes time comparable to the forward evaluation of the expected energy. Then, we optimize the iPEPS using a quasi-Newton L-BFGS algorithm [83] with an automatically computed gradient. One quickly reaches an optimum after a few hundreds of function and gradient evaluations. Figure 4(a) shows the relative error in energy compared to extrapolated quantum Monte Carlo (QMC) results [95] for various bond dimensions. The accuracy of the ground-state energy is comparable to state-of-the-art results [34,35], which were shown to be more accurate than imaginary-time projection-based simple and full update algorithms [29–33] [96]. Note that both the ansatz in Ref. [35] and our ansatz contain only half of the variational parameters as the one in Ref. [34], so the energy results are slightly higher than in Ref. [34] at $D = 2, 3$. However, for larger bond dimensions $D = 4, 5, 6, 7$, our calculation reaches the lowest variational energy for the infinite square lattice Heisenberg model. Figure 4(b) shows the staggered magnetization measured on the optimized state,

which approaches the extrapolated QMC results at larger bond dimensions.

To obtain results for bond dimension $D > 4$, we need to employ either the checkpoint technique in Sec. III B or the fixed-point iteration in Sec. III C to keep the memory budget low enough to fit into a single Nvidia P100 GPU card with 12G memory. It is rather encouraging that with moderate effort, one can reach the state-of-the-art performance in variationally optimizing iPEPS [34,35]. The success is also a nontrivial demonstration that one can indeed stabilize reverse-mode automatic differentiation for linear algebra operations appearing in scientific computation [47].

We note that the present approach applies as well to finite systems or problems with larger unit cells, more complex Hamiltonians [97–100], and more sophisticated contraction schemes with improved efficiency [94], which is promising to deliver new physical results to quantum many-body problems.

V. DISCUSSIONS

Computing the gradient via automatic differentiation significantly boosts the power of existing tensor network algorithms. Researchers can focus on the core tensor network contraction algorithms without worrying about the tedious gradient calculations. The computational complexity of automatic differentiation is the same as the forward contraction of the tensor networks.

Besides greatly reducing human efforts, the automatic differentiation approach also computes a slightly different gradient than Refs. [34,35]. The present approach computes the numerically exact gradient of an approximated energy density via automatic differentiation, while Refs. [34,35] first derive analytical expression of the energy gradient as infinite tensor networks and then contract these networks approximately to obtain an approximated gradient. Thus, the two approaches differentiate the approximation and approximate the derivative, respectively [44]. Other than the general recommendation of Ref. [44], we find that differentiating through approximated tensor network contraction can be advantageous for infinite systems whose analytical derivative is complicated to derive and approximate.

In this paper, we have focused on the high-level applications of automatic differentiation that differentiates through the whole contraction algorithms for optimizing tensor networks and computing physical observables. The same techniques are also applicable to low-level cases such as finding optimal truncation bases or variational transformation of tensor networks [64]. Moreover, besides the optimization of the expected energy of quantum problems, the approach is also relevant to variational contraction of tensor networks [2,101]. We expect that differentiable programming techniques will become an integral part of the standard tensor network toolbox.

A bonus of implementing tensor network programs using deep learning frameworks [51,53–55] is that one can readily enjoy the GPU acceleration. The calculations of this work were done with a single GPU card. Pushing this line of research further, we envision that it will be rewarding to deploy tensor network algorithms on emerging specialized hardware on a larger scale.

Finally, it is useful to comment on the difference of automatic differentiation for tensor networks and neural networks. Typical neural network architectures do not involve sophisticated linear algebra operations. However, with the development of tensorized neural networks [102] and applications of various tensor networks to machine learning problems [10–14,103], the boundary between the two classes of networks is blurred. Thus, results presented in this paper would also be relevant to tensor network machine learning applications when one moves to more sophisticated contraction schemes.

ACKNOWLEDGMENTS

We are grateful to Pan Zhang, Zhi-Yuan Xie, Wei Li, Ling Wang, Dian Wu, Xiu-Zhe Luo, Shuo-Hui Li, Song Cheng, Jing Chen, Xi Dai, Anders Sandvik, and Frank Verstraete for useful discussions. We thank Philippe Corboz and Laurens Vanderstraeten for providing the reference data shown in Fig. 4. The authors are supported by the National Key Research and Development Project of China Grants No. 2017YFA0302901 and No. 2016YFA0302400, the National Natural Science Foundation of China Grants No. 11888101 and No. 11774398, and the Strategic Priority Research Program of Chinese Academy of Sciences Grant No. XDB28000000.

[1] R. Orús, *A Practical Introduction to Tensor Networks: Matrix Product States and Projected Entangled Pair States*, Ann. Phys. (Amsterdam) **349**, 117 (2014).
[2] J. Haegeman and F. Verstraete, *Diagonalizing Transfer Matrices and Matrix Product Operators: A Medley of Exact and Computational Methods*, Annu. Rev. Condens. Matter Phys. **8**, 355 (2017).
[3] R. Orús, *Tensor Networks for Complex Quantum Systems*, Nat. Rev. Phys. https://doi.org/10.1038/s42254-019-0086-7 (2019).
[4] I. Markov and Y. Shi, *Simulating Quantum Computation by Contracting Tensor Networks*, SIAM J. Comput. **38**, 963 (2008).
[5] I. Arad and Z. Landau, *Quantum Computation and the Evaluation of Tensor Networks*, SIAM J. Comput. **39**, 3089 (2010).
[6] I. H. Kim and B. Swingle, *Robust Entanglement Renormalization on a Noisy Quantum Computer*, arXiv:1711.07500.
[7] W. Huggins, P. Patel, K. B. Whaley, and E. M. Stoudenmire, *Towards Quantum Machine Learning with Tensor Networks*, Quantum Sci. Technol. **4**, 024001 (2019).
[8] A. J. Ferris and D. Poulin, *Tensor Networks and Quantum Error Correction*, Phys. Rev. Lett. **113**, 030501 (2014).

[9] S. Bravyi, M. Suchara, and A. Vargo, *Efficient Algorithms for Maximum Likelihood Decoding in the Surface Code*, Phys. Rev. A **90**, 032326 (2014).

[10] E. M. Stoudenmire and D. J. Schwab, *Supervised Learning with Tensor Networks*, Adv. Neural Inf. Process. Syst. **29**, 4799 (2016).

[11] E. M. Stoudenmire, *Learning Relevant Features of Data with Multi-scale Tensor Networks*, Quantum Sci. Technol. **3**, 034003 (2018).

[12] Z.-Yu Han, J. Wang, H. Fan, L. Wang, and P. Zhang, *Unsupervised Generative Modeling Using Matrix Product States*, Phys. Rev. X **8**, 031012 (2018).

[13] S. Cheng, L. Wang, T. Xiang, and P. Zhang, *Tree Tensor Networks for Generative Modeling*, Phys. Rev. B **99**, 155131 (2019).

[14] J. Stokes and J. Terilla, *Probabilistic Modeling with Matrix Product States*, arXiv:1902.06888v1.

[15] A. J. Gallego and R. Orús, *Language Design and Renormalization*, arXiv:1708.01525.

[16] V. Pestun and Y. Vlassopoulos, *Tensor Network Language Model*, arXiv:1710.10248.

[17] F. Verstraete and J. I. Cirac, *Continuous Matrix Product States for Quantum Fields*, Phys. Rev. Lett. **104**, 190405 (2010).

[18] J. Haegeman, T. J. Osborne, H. Verschelde, and F. Verstraete, *Entanglement Renormalization for Quantum Fields*, Phys. Rev. Lett. **110**, 100402 (2013).

[19] Q. Hu, A. Franco-Rubio, and G. Vidal, *Continuous Tensor Network Renormalization for Quantum Fields*, arXiv:1809.05176.

[20] A. Tilloy and J. I. Cirac, *Continuous Tensor Network States for Quantum Fields*, Phys. Rev. X **9**, 021040 (2019).

[21] B. Swingle, *Entanglement Renormalization and Holography*, Phys. Rev. D **86**, 065007 (2012).

[22] P. Hayden, S. Nezami, X. L. Qi, N. Thomas, M. Walter, and Z. Yang, *Holographic Duality from Random Tensor Networks*, J. High Energy Phys. 11 (2016) 009.

[23] S. R. White, *Density Matrix Formulation for Quantum Renormalization Groups*, Phys. Rev. Lett. **69**, 2863 (1992).

[24] G. Vidal, *Classical Simulation of Infinite-Size Quantum Lattice Systems in One Spatial Dimension*, Phys. Rev. Lett. **98**, 070201 (2007).

[25] U. Schollwöck, *The Density-Matrix Renormalization Group in the Age of Matrix Product States*, Ann. Phys. (Amsterdam) **326**, 96 (2011).

[26] E. M. Stoudenmire and S. R. White, *Studying Two Dimensional Systems with the Density Matrix Renormalization Group*, Annu. Rev. Condens. Matter Phys. **3**, 111 (2011).

[27] Z. Landau, U. Vazirani, and T. Vidick, *A Polynomial Time Algorithm for the Ground State of One-Dimensional Gapped Local Hamiltonians*, Nat. Phys. **11**, 566 (2015).

[28] V. Zauner-Stauber, L. Vanderstraeten, M. T. Fishman, F. Verstraete, and J. Haegeman,*Variational Optimization Algorithms for Uniform Matrix Product States*, Phys. Rev. B **97**, 045145 (2018).

[29] H. C. Jiang, Z. Y. Weng, and T. Xiang, *Accurate Determination of Tensor Network State of Quantum Lattice Models in Two Dimensions*, Phys. Rev. Lett. **101**, 090603 (2008).

[30] J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac, *Classical Simulation of Infinite-Size Quantum Lattice Systems in Two Spatial Dimensions*, Phys. Rev. Lett. **101**, 250602 (2008).

[31] P. Corboz, R. Orús, B. Bauer, and G. Vidal, *Simulation of Strongly Correlated Fermions in Two Spatial Dimensions with Fermionic Projected Entangled-Pair States*, Phys. Rev. B **81**, 165104 (2010).

[32] H. H. Zhao, Z. Y. Xie, Q. N. Chen, Z. C. Wei, J. W. Cai, and T. Xiang, *Renormalization of Tensor-Network States*, Phys. Rev. B **81**, 174411 (2010).

[33] H. N. Phien, J. A. Bengua, H. D. Tuan, P. Corboz, and R. Orús, *Infinite Projected Entangled Pair States Algorithm Improved: Fast Full Update and Gauge Fixing*, Phys. Rev. B **92**, 035142 (2015).

[34] P. Corboz, *Variational Optimization with Infinite Projected Entangled-Pair States*, Phys. Rev. B **94**, 035133 (2016).

[35] L. Vanderstraeten, J. Haegeman, P. Corboz, and F. Verstraete, *Gradient Methods for Variational Optimization of Projected Entangled-Pair States*, Phys. Rev. B **94**, 155123 (2016).

[36] D. Poilblanc and M. Mambrini, *Quantum Critical Phase with Infinite Projected Entangled Paired States*, Phys. Rev. B **96**, 014414 (2017).

[37] J.-Y. Chen, L. Vanderstraeten, S. Capponi, and D. Poilblanc, *Non-Abelian Chiral Spin Liquid in a Quantum Antiferromagnet Revealed by an iPEPS Study*, Phys. Rev. B **98**, 184409 (2018).

[38] L. Wang, Y.-J. Kao, and A. W. Sandvik, *Plaquette Renormalization Scheme for Tensor Network States*, Phys. Rev. E **83**, 056703 (2011).

[39] See https://github.com/wangleiphy/tensorgrad for code implementation in PyTorch.

[40] M. Bartholomew-Biggs, S. Brown, B. Christianson, and L. Dixon, *Automatic Differentiation of Algorithms*, J. Comput. Appl. Math. **124**, 171 (2000).

[41] W. Baur and V. Strassen, *The Complexity of Partial Derivatives*, Theor. Comput. Sci. **22**, 317 (1983).

[42] A. Griewank, *On Automatic Differentiation*, in *Math. Program. Recent Dev. Appl.* (Kluwer Academic Publishers, Dordrecht, The Netherlands, 1989), pp. 83–108.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-Propagating Errors*, Nature (London) **323**, 533 (1986).

[44] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic Differentiation in Machine Learning: A Survey*, J. Mach. Learn. **18**, 1 (2015).

[45] N. Leung, M. Abdelhafez, J. Koch, and D. Schuster, *Speedup for Quantum Optimal Control from Automatic Differentiation Based on Graphics Processing Units*, Phys. Rev. A **95**, 042318 (2017).

[46] S. Sorella and L. Capriotti, *Algorithmic Differentiation and the Calculation of Forces by Quantum Monte Carlo*, J. Chem. Phys. **133** (2010).

[47] T. Tamayo-Mendoza, C. Kreisbeck, R. Lindh, and A. Aspuru-Guzik, *Automatic Differentiation in Quantum Chemistry with Applications to Fully Variational Hartree-Fock*, ACS Cent. Sci. **4**, 559 (2018).

[48] As noted in Ref. [49], one will need tensor calculus as a precise language to present automatic differentiation, even though the object being differentiated is a matrix.
[49] S. Laue, M. Mitterreiter, and J. Giesen, *Computing Higher Order Derivatives of Matrix and Tensor Expressions*, Neural Inf. Proc. Syst. **31**, 2750 (2018).
[50] A. Griewank and A. Walther, *Evaluating Derivatives* (Society for Industrial and Applied Mathematics, 2008).
[51] M. Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, arXiv:1603.04467.
[52] D. Maclaurin, D. Duvenaud, and R. P. Adams, *Autograd: Effortless Gradients in Numpy*, in *ICML 2015 AutoML Workshop* (2015).
[53] A. Paszke, G. Chanan, Z. Lin, S. Gross, E. Yang, L. Antiga, and Z. Devito, *Automatic Differentiation in PyTorch*, *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*.
[54] M. Johnson, R. Frostig, and C. Leary, *Compiling Machine Learning Programs via High-Level Tracing*, in *SysML* (2018).
[55] M. Innes, *Don't Unroll Adjoint: Differentiating SSA-Form Programs*, arXiv:1810.07951.
[56] S. Kourtis, C. Chamon, E. R. Mucciolo, and A. E. Ruckenstein, *Fast Counting with Tensor Networks*, arXiv:1805.00475.
[57] B. Villalonga, S. Boixo, B. Nelson, E. Rieffel, R. Biswas, and S. Mandr, *A Flexible High-Performance Simulator for the Verification and Benchmarking of Quantum Circuits Implemented on Real Hardware Benjamin*, arXiv:1811.09599v1.
[58] M. Levin and C. P. Nave, *Tensor Renormalization Group Approach to Two-Dimensional Classical Lattice Models*, Phys. Rev. Lett. **99**, 120601 (2007).
[59] Z. Y. Xie, H. C. Jiang, Q. N. Chen, Z. Y. Weng, and T. Xiang, *Second Renormalization of Tensor-Network States*, Phys. Rev. Lett. **103**, 160601 (2009).
[60] Z.-C. Gu and X.-G. Wen, *Tensor-Entanglement-Filtering Renormalization Approach and Symmetry-Protected Topological Order*, Phys. Rev. B **80**, 155131 (2009).
[61] Z. Y. Xie, J. Chen, M. P. Qin, J. W. Zhu, L. P. Yang, and T. Xiang, *Coarse-Graining Renormalization by Higher-Order Singular Value Decomposition*, Phys. Rev. B **86**, 045139 (2012).
[62] Z. Y. Xie, J. Chen, J. F. Yu, X. Kong, B. Normand, and T. Xiang, *Tensor Renormalization of Quantum Many-Body Systems Using Projected Entangled Simplex States*, Phys. Rev. X **4**, 011025 (2014).
[63] G. Evenbly and G. Vidal, *Tensor Network Renormalization*, Phys. Rev. Lett. **115**, 180405 (2015).
[64] S. Yang, Z.-C. Gu, and X.-G. Wen, *Loop Optimization for Tensor Network Renormalization*, Phys. Rev. Lett. **118**, 110504 (2017).
[65] T. Nishino and K. Okunishi, *Corner Transfer Matrix Renormalization Group Method*, J. Phys. Soc. Jpn. **65**, 891 (1996).
[66] R. Orús, *Exploring Corner Transfer Matrices and Corner Tensors for the Classical Simulation of Quantum Lattice Systems*, Phys. Rev. B **85**, 205117 (2012).
[67] P. Corboz, T. M. Rice, and M. Troyer, *Competing States in the t-J Model: Uniform d-Wave State versus Stripe State Philippe*, Phys. Rev. Lett. **113**, 046402 (2014).
[68] R. Orus and G. Vidal, *Infinite Time-Evolving Block Decimation Algorithm Beyond Unitary Evolution*, Phys. Rev. B **78**, 155117 (2008).
[69] M. Giles, *An Extended Collection of Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation*, Technical Report 2008, https://people.maths.ox.ac.uk/gilesm/files/NA-08-01.pdf.
[70] J. Townsend, *Differentiating the Singular Value Decomposition*, Technical Report 2016, https://j-towns.github.io/papers/svd-derivative.pdf.
[71] M. Seeger, A. Hetzel, Z. Dai, E. Meissner, and N. D. Lawrence, *Auto-Differentiating Linear Algebra*, arXiv:1710.08717.
[72] R. P. Feynman, *Forces on Molecules*, Phys. Rev. **56**, 340 (1939).
[73] H. Fujita, Y. O. Nakagawa, S. Sugiura, and M. Oshikawa, *Construction of Hamiltonians by Supervised Learning of Energy and Entanglement Spectra*, Phys. Rev. B **97**, 075114 (2018).
[74] R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods* (Cambridge University Press, Cambridge, England, 2004).
[75] The cases where one has exact degeneracy between occupied and unoccupied eigenstates is itself a singular case. One should include or exclude all degenerated states in these cases.
[76] F. Pollmann, A. M. Turner, E. Berg, and M. Oshikawa, *Entanglement Spectrum of a Topological Phase in One Dimension*, Phys. Rev. B **81**, 064439 (2010).
[77] E. Efrati, Z. Wang, A. Kolan, and L. P. Kadanoff, *Real-Space Renormalization in Statistical Mechanics*, Rev. Mod. Phys. **86**, 647 (2014).
[78] One can also require the diagonal of the R matrix to be positive to fix the redundant gauge d.o.f. This can be easily implemented on top of the existing QR libraries with backward support.
[79] T. Chen, B. Xu, C. Zhang, and C. Guestrin, *Training Deep Nets with Sublinear Memory Cost*, arXiv:1604.06174.
[80] B. Christianson, *Reverse Accumulation and Attractive Fixed Points*, Optim. Methods Software **3**, 311 (1994).
[81] M. T. Fishman, L. Vanderstraeten, V. Zauner-Stauber, J. Haegeman, and F. Verstraete, *Faster Methods for Contracting Infinite Two-Dimensional Tensor Networks*, Phys. Rev. B **98**, 235148 (2018).
[82] A related example is backward through an iterative solver for the dominant eigenstates x^* of a matrix A. In the forward process, one may use the Krylov space method such as Lanczos or Arnoldi iterations. For the backward pass, one can leverage the fact that the dominant eigenvector is the fixed point of the power iteration $f(x, A) = Ax/\|Ax\|$. Therefore, one can use Eq. (7) to propagate $\overline{x^*}$ back to $\bar{A}$. Note that the forward and the backward iteration functions are decoupled; thus, the backward function does not need to be the same as the forward pass, as long as it ensures x^* is a fixed point.
[83] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. (Springer, New York, 2006).

[84] B. A. Pearlmutter, *Fast Exact Multiplication by the Hessian*, Neural Comput. **6**, 147 (1994).
[85] See https://github.com/pytorch/pytorch.
[86] See https://github.com/HIPS/autograd.
[87] See https://github.com/tensorflow/tensorflow.
[88] See https://github.com/google/jax.
[89] See https://github.com/FluxML/Zygote.jl.
[90] J. Chen, H.-J. Liao, H.-D. Xie, X.-J. Han, R.-Z. Huang, S. Cheng, Z.-C. Wei, Z.-Y. Xie, and T. Xiang, *Phase Transition of the Q-State Clock Model: Duality and Tensor Renormalization*, Chin. Phys. Lett. **34**, 050503 (2017).
[91] W. Li, J. von Delft, and T. Xiang, *Efficient Simulation of Infinite Tree Tensor Network States on the Bethe Lattice*, Phys. Rev. B **86**, 195137 (2012).
[92] L. Vanderstraeten, M. Mariën, F. Verstraete, and J. Haegeman, *Excitations and the Tangent Space of Projected Entangled-Pair States*, Phys. Rev. B **92**, 201111(R) (2015).
[93] L. Onsager, *Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition*, Phys. Rev. **65**, 117 (1944).
[94] Z. Y. Xie, H. J. Liao, R. Z. Huang, H. D. Xie, J. Chen, Z. Y. Liu, and T. Xiang, *Optimized Contraction Scheme for Tensor-Network States*, Phys. Rev. B **96**, 045128 (2017).
[95] A. W. Sandvik, *Computational studies of quantum spin systems*, AIP Conf. Proc. **1297**, 135 (2010).
[96] Although it was expected that the full update method will, in principle, reach the same accuracy as the variational method within the error of Trotter-Suzuki decomposition, it would require optimizing the iPEPS tensors in a globally optimal way.
[97] P. Corboz, S. R. White, G. Vidal, and M. Troyer, *Stripes in the Two-Dimensional t-J Model with Infinite Projected Entangled-Pair States*, Phys. Rev. B **84**, 041108(R) (2011).
[98] H. J. Liao, Z. Y. Xie, J. Chen, Z. Y. Liu, H. D. Xie, R. Z. Huang, B. Normand, and T. Xiang, *Gapless Spin-Liquid Ground State in the $S = 1/2$ Kagome Antiferromagnet*, Phys. Rev. Lett. **118**, 137202 (2017).
[99] C.-Y. Lee, B. Normand, and Y.-J. Kao, *Gapless Spin Liquid in the Kagome Heisenberg Antiferromagnet with Dzyaloshinskii-Moriya Interactions*, Phys. Rev. B **98**, 224414 (2018).
[100] R. Haghshenas, S.-S. Gong, and D. N. Sheng, *An iPEPS Study of Kagome Heisenberg Model with Chiral Interaction: A Single-Layer Tensor-Network Algorithm*, Phys. Rev. B **99**, 174423 (2019).
[101] L. Vanderstraeten, B. Vanhecke, and F. Verstraete, *Residual Entropies for Three-Dimensional Frustrated Spin Systems with Tensor Networks*, Phys. Rev. E **98**, 042145 (2018).
[102] C. Ionescu, O. Vantzos, and C. Sminchisescu, *Training Deep Networks with Structured Layers by Matrix Backpropagation*, Proc. IEEE Int. Conf. Comput. Vis. 2965 (2015).
[103] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, *Tensorizing Neural Networks*, Adv. Neural Inf. Process. Syst. 442 (2015).