Variational optimization of ground-state iPEPS via automatic differentiation

Dimitra I. Nikolaidou

München 2024

Variational optimization of ground-state iPEPS via automatic differentiation

Dimitra I. Nikolaidou

Master's thesis
Faculty of Physics
Chair of Theoretical Condensed-Matter Physics
Ludwig–Maximilians–Universität
München

Dimitra I. Nikolaidou

München, 2024

Supervisor and First examiner: Prof. Dr. Jan von Delft
Second examiner: Prof. Dr. Frank Pollmann

Abstract

Simulations of quantum many-body systems are invaluable for gaining insight into the nature of the various phases of strongly correlated matter, the nature of the excitations which emerge in many-body systems, as well as their dynamics. The continuously growing field of tensor network methods involves a variety of algorithms which provide a framework to simulate bosonic and fermionic systems of many particles at the strongly interacting regime. Specifically, tensor networks are particularly successful in representing ground states of effective Hamiltonians used in quantum many-body physics. In order to take advantage of this fact, accurate algorithms must be devised to obtain the tensors which best represent the true ground state(s) of a system. A primitive algorithm for ground-state search of 2D tensor networks is the simple update algorithm, based on imaginary time evolution, which is accurate for tree-like networks but becomes inaccurate for tensor networks with loops. Numerous improvements to this algorithm exist, including various algorithms based on variational optimization of tensor networks. Among these, the focus of this thesis is gradient-based optimization, which uses gradient information of the variational energy in order to optimize over the tensor network parameters and achieve the lowest possible energy.

This thesis focused on the development of an implementation of reverse-mode Automatic Differentiation (AD), a method that allows for precise and efficient gradient evaluation. The energy gradient information obtained in this way was used for gradient-based optimization of infinite Projected Entangled Pair States (iPEPS), in the context of ground-state search for various quantum lattice models. The QSpace library for tensor network operations was chosen, which allows for the implementation of abelian and non-abelian symmetries on tensor networks. An efficient implementation of AD for the Corner Transfer Matrix scheme was investigated in detail and applied to the 2D Heisenberg model, a 2D free fermionic model, and the Hubbard model at half-filling, all with nearest-neighbor interactions. These benchmarks demonstrated the performance of the optimization procedure and quantified the improvement over the simple update algorithm. For the case of nearest-neighbor Hubbard model, the potential of the recently proposed Belief Propagation gauging was investigated as a way to improve over the usual simple update algorithm for pre-training the tensor network before the optimization. Finally, the Hubbard model with next-nearest neighbor hopping amplitude $t' = 0.25$ at finite doping is considered, and variational optimization is applied to a U(1) state with period eight, and a uniform, SU(2) symmetric state.

Acknowledgments

I would first like to thank Prof. Dr. Jan von Delft for giving me the opportunity to work on an interesting and modern topic, and for passing on his enthusiasm for the field of tensor networks. I feel very lucky to be included in his group, as he created a positive atmosphere and consistently provided encouragement, care, and careful guidance to me and the rest of the group members.

I would also like to express my appreciation for the support I received from Changkai Zhang, who was always there to answer my questions with a particular clarity of thought and with strategic suggestions. Additionally, I am thankful for the time he devoted to his own master's project, as this thesis partly builds upon his, and for the nicely drawn diagrams which he shared with me. I am also grateful for the assistance provided by Dr. Markus Scheb, who was always eager to share with me his technical expertise and useful advice when I was stuck.

I must express my thanks to two individuals I have never met, yet whose contributions were vital to the completion of this project. First, thanks to Jheng-Wei Li for sharing with me the backbone of a code which I used, crucial also as a guide for me to take the first steps on this project. Second, a special thanks should also go to Andreas Weichselbaum for the hours he spent on building the QSpace library used in this project.

Finally, I wish to thank Leonardo Bezzo for proofreading this manuscript, as well as for always being at my side the past year, with meaningful advice at moments of doubt, engaging discussions and practical help, to matters of physics or otherwise.

Contents

Introduction

The ground-state problem, concerning the identification of the lowest energy of a system at zero temperature and its associated state(s), is one of the most important questions in quantum many-body physics, with no known universally applicable solution. It is physically and technologically a highly relevant question to ask, since quantum many-body systems, especially at the strongly interacting regime, might reveal previously unknown, exotic phases at low temperatures, whose realization might provide useful technological applications.

Interactions between individual quantum particles are responsible for entanglement generation; strong, local interactions tend to create locally strongly entangled states. The amount and distribution of entanglement across the system, which are controlled by the Hamiltonian, determine the system's properties at zero temperature. Different parameter regimes of the Hamiltonian might correspond to distinct phases with diametrically different properties. A notable example is the Mott transition of strongly interacting electronic systems, which is a shift from a metallic to an insulating phase controlled by the interaction strength. Strongly interacting systems can be modeled via variational wavefunctions, classes of quantum states which depend on some variational parameters, and are considered to describe well the real ground states at some parameter regimes. Tensor network variational ansätze provide a natural framework for the modeling of states with strong, local correlations. Their usefulness comes into play when considering that due to the locality of physical interactions, "physical" entangled states are restricted to a small subspace of the total, exponentially large, Hilbert space. Due to this property, tensor networks can provide accurate variational descriptions of ground states even with a relatively small number of parameters.

A large portion of tensor network applications is concerned with the ground-state search for quantum many-body systems. One method of approaching the problem is through the variational optimization of tensor network ansätze for the identification of the minimum of the corresponding variational energy. A straightforward way to obtain the minimum over the parameter space is to adopt a gradient-based optimization procedure. There are several methods developed for this purpose outside the context of tensor networks, which can be applied directly to them. The main obstacle is the calculation of the energy gradient. A particularly promising technique in this respect is that of Automatic Differentiation (AD), a method to obtain gradient values efficiently up to machine precision, which is widely used in machine learning. AD was originally proposed for use in tensor networks in [LLWX19],

where promising benchmarks were provided for second-derivative calculations of the partition function of the Ising model, and for ground-state search of the 2D Heisenberg model. It quickly became a standard method for tensor network optimization, with applications to several problems, primarily regarding ground-state search [NWR+24] [LS23] [WR24].

This year-long thesis project had as its main scope the implementation of such an AD-based optimization of tensor networks. Although several libraries for this already exist (TensorFlow, Jax, PyTorch, and more), in this thesis the implementation was done from scratch[1] for the QSpace library [Wei24]. QSpace offers the ability to easily implement abelian and non-abelian symmetries for tensor network calculations, which results in more efficient computations. The technique was benchmarked for the infinite Projected Entangled Pair States, and the performance of the AD implementation was studied in the context of variational optimization for several quantum lattice models.

[1] For this, the contribution of Jheng-Wei Li needs to be stated, who provided a first version of the code for some basic AD operations used in tensor networks.

Chapter 1

Automatic differentiation for tensor networks

Automatic differentiation (AD) is a technique to calculate derivative values of arbitrary differentiable functions expressed as computer programs [BBCD00]. It is a simple method based on the chain rule of differentiation, but it is very powerful, widely preferred for non-linear optimization problems over other methods such as numerical and symbolic differentiation. The term *automatic* highlights the fact that there is no need for "manual" calculation of each gradient. Instead, only the gradients of functions of a few fundamental operations need to be constructed, and the full gradient is obtained from a propagation through the chain rule. The key advantages of AD are its mathematically guaranteed efficiency, flexibility in terms of applications, and the ability to calculate gradients up to machine precision. The last feature distinguishes it from numerical methods of gradient calculation, such as the finite-differences method, which inevitably introduce errors.

This chapter aims at introducing the concepts of AD that were found to be useful for its application to tensor networks. It also gives an introduction on some important tensor network concepts and an exemplary application of AD on tensor networks, which was implemented for this thesis. The chapter is divided as follows. The basic idea of AD is described in Sec. 1.1, where forward- and reverse-mode AD are explained in the context of functions of real variables. The generalization to complex-variable functions is also given. This is followed by Sec. 1.2, where the validity of tensor networks as frameworks for simulating quantum many-body ground states is discussed, together with some basic concepts of tensor networks. In Sec. 1.3.1 the Corner Transfer Matrix Renormalization Group is described, which is followed by Sec. 1.3.2 on how to efficiently implement reverse-mode AD in it.

1.1 Basics on automatic differentiation (AD)

Suppose that we have a real function $\boldsymbol{f} : \mathbb{R}^{m_0} \to \mathbb{R}^{m_N}$, differentiable at a point $\boldsymbol{x_0} \in \mathbb{R}^{m_0}$ where we want to evaluate the Jacobian

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{m_0}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{m_N}}{\partial x_1} & \cdots & \frac{\partial f_{m_N}}{\partial x_{m_0}} \end{pmatrix} \tag{1.1}$$

Suppose also that we are equipped with an algorithm for the computation of $\boldsymbol{f}(\boldsymbol{x_0})$. The computer program for this calculation is assumed to consist of smaller steps and can be depicted as a directed, acyclic graph. The acyclicity implies that, in case of writing-over variables, which is common in function evaluations, the graph is written in such a way that the overwritten values are stored as new variables. An example of the types of graphs that we are interested in in this thesis can be seen in Fig. 1.1a. Since it is common to work with vectors instead of elements of vectors, it is convenient to write the same graph in a way that each layer is compressed to one vertex which corresponds to a whole vector of variables independent to each other, as shown, for example, in Fig. 1.1b. For the rest of the section we will work at this vectorial level.

The graph is composed of an input layer plus N vertices labeled by $i \in \{1, 2, ..., N\}$ and each vertex $\boldsymbol{v^{(i)}}$ is related to preceding connected nodes via some function

$$\boldsymbol{\phi^{(i)}} : \mathbb{R}^{l_i} \to \mathbb{R}^{m_i} \tag{1.2}$$

which is assumed to be continuously differentiable in $\mathbb{R}^{l_i}$. The input node corresponds to the initial condition:

$$\boldsymbol{v^{(0)}} = \boldsymbol{x_0} \tag{1.3}$$

and the last node to the function value:

$$\boldsymbol{v^{(N)}} = \boldsymbol{f}(\boldsymbol{x_0}) \tag{1.4}$$

For each vertex there exist a set of nodes called *parent/child nodes* which are the preceding/succeeding nodes *directly* connected to it. The following notation is adopted:

$$\boxed{\begin{array}{c} P_i : \text{ set with elements the parent nodes of vertex } \boldsymbol{v^{(i)}} \\ C_i\text{: set with elements the child nodes of vertex } \boldsymbol{v^{(i)}} \end{array}} \tag{1.5}$$

This means that at step i the following mapping occurs:

$$P_i = \{\boldsymbol{y_1}, ..., \boldsymbol{y_{l_i}}\}|_{\boldsymbol{x_0}} \to \boldsymbol{v^{(i)}} = \boldsymbol{\phi^{(i)}}(\boldsymbol{y_1}, ..., \boldsymbol{y_{l_i}})\big|_{\boldsymbol{x_0}} \tag{1.6}$$

For example, in Fig. 1.1 the parent nodes of vertex $\boldsymbol{v^{(2)}}$ are $P_2 = \{\boldsymbol{v^{(0)}}, \boldsymbol{v^{(1)}}\}$ and for $\boldsymbol{v^{(3)}}$ is $P_3 = \{\boldsymbol{v^{(2)}}\}$. Their child nodes are, respectively, $C_2 = \{\boldsymbol{v^{(3)}}, \boldsymbol{v^{(4)}}\}$ and $C_3 = \{\boldsymbol{v^{(4)}}\}$.

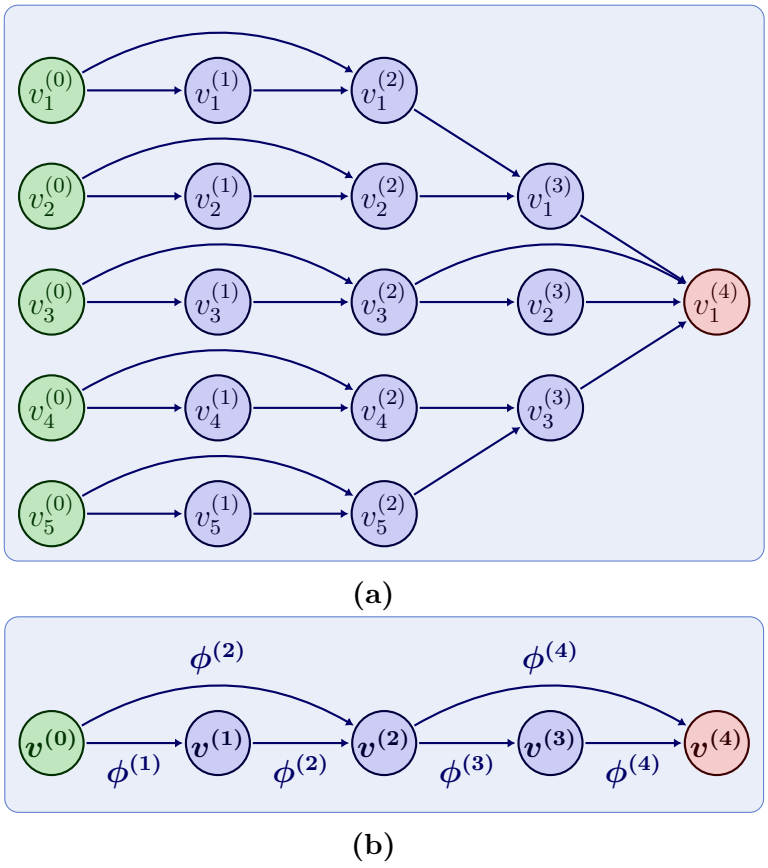


Figure 1.1: (a) Graph showing a function evaluation procedure in the form of an acyclic, directed graph, broken down to multiple layers of nodes. (b) Equivalent of the graph in (a) but choosing vectorial representation for the vertices, indicating also the corresponding mappings $\boldsymbol{\phi}^{(i)}$ from previous layers to layer i.

Another useful notation is that for the tangents and gradients of intermediate steps. I will make use of the following definitions which are common in the literature[1]:

$$\begin{aligned} \dot{\boldsymbol{y}} &:= \frac{\partial \boldsymbol{y}}{\partial x_k} = \left(\frac{\partial y_1}{\partial x_k}, \frac{\partial y_2}{\partial x_k}, ..., \frac{\partial y_l}{\partial x_k} \right)^T, \qquad k \in \{1, 2, ..., m_0\} \\ \bar{\boldsymbol{y}} &:= \frac{\partial f_k}{\partial \boldsymbol{y}} = \left(\frac{\partial f_k}{\partial y_1}, \frac{\partial f_k}{\partial y_2}, ..., \frac{\partial f_k}{\partial y_l} \right), \qquad k \in \{1, 2, ..., m_N\} \end{aligned} \tag{1.7}$$

When $\boldsymbol{\phi}^{(i)}$ are chosen to be elementary operations used in the program, the corresponding graph is called the *computational graph* [Bau74]. Such elementary operations are called

[1]This notation is a bit confusing since the derivatives depend on k while $\dot{\boldsymbol{y}}$ and $\bar{\boldsymbol{y}}$ don't encode this information. However, since it is a well-established notation in the AD literature, it is adopted in this manner. Later it will become more clear that this notation is justified by the fact that one forward or reverse pass of AD can only refer to a constant value of k.

primitives in the AD literature. What is considered primitive depends on the user and the application. As we will see in the next section, for tensor network applications these might be commonly used functions for which analytical expressions for the derivatives are known, such as addition, contraction, singular value decomposition, etc.

The AD calculation of the gradients is not a symbolic one, and therefore in practice the vertices take specific numerical values. However, in order to develop the AD method, $\boldsymbol{v^{(i)}}$ are seen as variables which directly or indirectly depend on the input vector. Using the chain rule, an element of the Jacobian (1.1) is given by:

$$\frac{\partial f_i}{\partial x_j} = \sum_{y \in P_N} \frac{\partial v_i^{(N)}}{\partial y} \frac{\partial y}{\partial x_j} \tag{1.8}$$

There are two mathematically equivalent ways to use this equation to calculate such elements, known as *forward-* and *reverse-mode* automatic differentiation. The term *forward* is used to indicate that the calculation of derivatives is happening in parallel with the function evaluation, contrary to the reverse mode where the function evaluation needs to be completed first (forward sweep) and then the gradient is computed in a way that scans backwards the computational graph (reverse sweep). Each method has its advantages, and for tensor network applications, the reverse mode is preferred. In order to understand the reason for this, a comparison of the two methods follows.

Forward mode

Suppose first that we would like to compute a specific *column* k of the Jacobian:

$$\dot{\boldsymbol{v}}^{(\boldsymbol{N})} = \left. \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_k} \right|_{\boldsymbol{x_0}} \tag{1.9}$$

In the forward-mode AD, at each step of the function evaluation the corresponding tangent is calculated in direction k. Specifically, functions that calculate the Jacobian-vector products of the primitives $\{\frac{\partial \boldsymbol{\phi}^{(j)}}{\partial \boldsymbol{y}} \dot{\boldsymbol{y}} \ \forall \boldsymbol{y} \in P_j\}$ need to be constructed, and by using the chain rule the result automatically propagates through all the parent nodes of $\boldsymbol{v^{(N)}}$. The tangents $\dot{\boldsymbol{y}}$ are known from previous steps of the chain rule, after initilizing $\dot{\boldsymbol{x}}_{\boldsymbol{0}}$. Fig. 1.2 shows the computation steps of the forward-mode AD. Note that the arrows of the chart show only the order of operations and are not related to the connectivity of the graph. In order to obtain the full Jacobian (1.1), this procedure must be repeated for all k.

Reverse mode

For the reverse mode, suppose first that we would like to compute a specific *row* k of the Jacobian at point $\boldsymbol{x_0}$:

$$\overline{\boldsymbol{x_0}} \coloneqq \boldsymbol{\nabla} f_k(\boldsymbol{x})|_{\boldsymbol{x_0}} = \left. \left(\frac{\partial f_k}{\partial x_1}, \frac{\partial f_k}{\partial x_2}, ..., \frac{\partial f_k}{\partial x_{m_0}} \right) \right|_{\boldsymbol{x_0}} \tag{1.10}$$

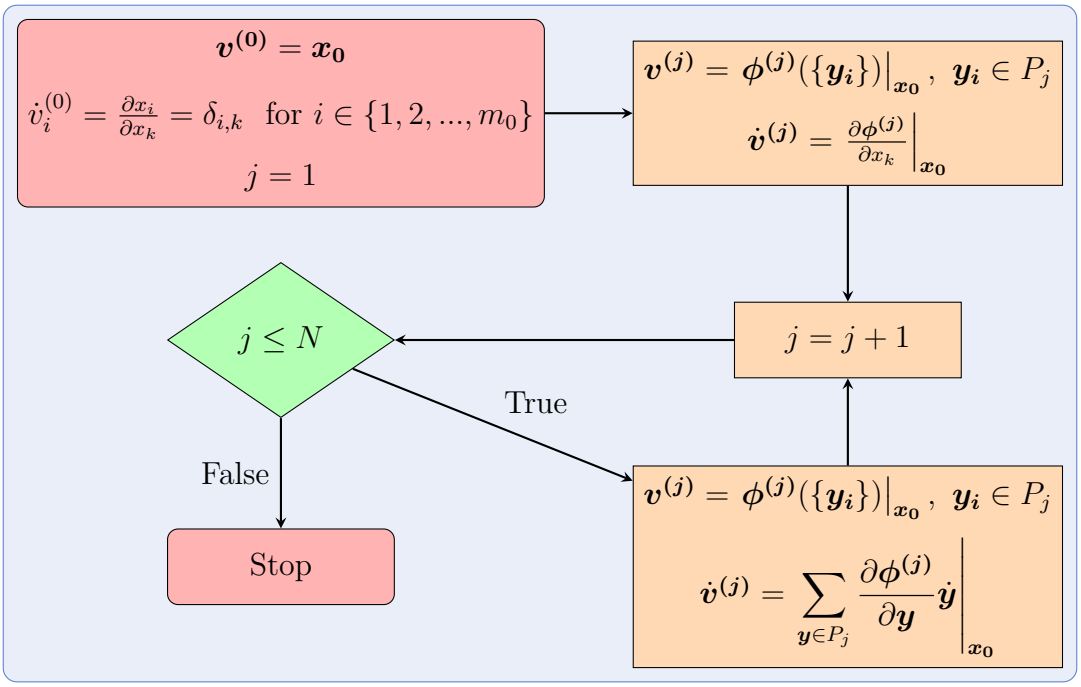


Figure 1.2: Schematic of the flow of computations in the forward-mode automatic differentiation routine.

For this calculation, after the function evaluation we start from the vector $\boldsymbol{v^{(N)}}$ made up of m_N independent variables, and initialize $\overline{\boldsymbol{v^{(N)}}}$. Then, by constructing the vector-Jacobian products of the primitives $\{\bar{\boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{v^{(j)}}} \; \forall \boldsymbol{y} \in C_j\}$, the chain rule can be used to propagate the gradient backward through all child nodes of $\boldsymbol{x_0}$. For this reason, reverse-mode AD is also referred to as *backpropagation*. Gradients $\bar{\boldsymbol{y}}$ are known from previous steps of the chain rule. In Fig. 1.3, the process of calculating Eq. (1.10) through reverse-mode automatic differentiation is illustrated. In order to obtain the full Jacobian (1.1) for all the m_N final variables, the procedure must be repeated for each element of $\boldsymbol{f}(\boldsymbol{x_0})$.

Computational complexity

From the above we can see that whenever $m_0 >> m_N$, i.e. the input dimension is much larger than the output dimension, the reverse mode is the most efficient choice for obtaining the full Jacobian. This is the case for applications in physics or in machine learning, where f corresponds to a scalar such as the energy or a cost function respectively, and therefore $m_N = 1$, while the inputs are large tensors or large vectors of data. On the contrary, when $m_0 << m_N$ the forward mode AD is the best option.

In general, both the forward and reverse modes are very efficient when $m_0 << m_N$ and $m_N << m_0$ correspondingly. Specifically, their algorithmic complexity is mathematically guaranteed to be of the same order as that of the function evaluation [GW08][Gri89].

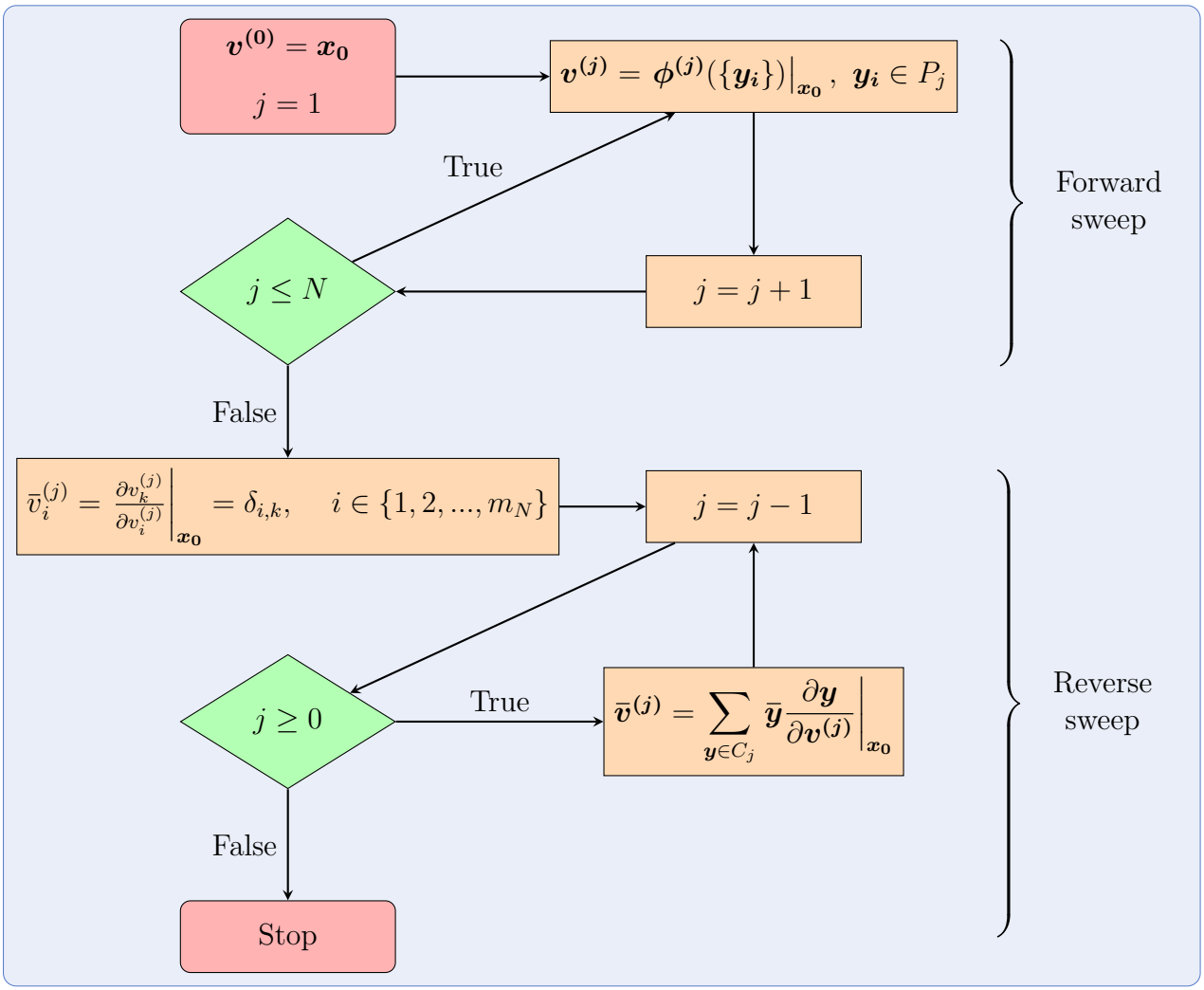


Figure 1.3: Schematic of the flow of computations in the reverse-mode automatic differentiation routine.

Memory cost

One of the biggest drawbacks of backpropagation compared to the forward-mode is its storage requirement. Notice that in order to calculate $\{\bar{\boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{v}^{(j)}} \; \forall \, \boldsymbol{y} \in C_j\}$ after the function evaluation, we need all the information from the computational graph: the values at each node and what operations $\boldsymbol{\phi^{(i)}}$ lead to these values. This means that all this information needs to be stored during the forward sweep. This, in tensor network applications that depend on fixed-point iterations, might quickly drive the memory requirements to the order of hundreds of GB. A common tool that partially remedies this is *checkpointing*. This consists of saving the computational graph at specific parts of the forward sweep (checkpoints), and during the reverse sweep to rerun small points of the forward sweep

that were not saved, in order to obtain the corresponding gradients via backpropagation on these smaller parts. After, the variables saved at for this reevaluation can be cleared from memory. This inevitably brings an increase in the computational cost. Specific examples are given in the following sections, in the context of the Corner Transfer Matrix algorithm. Another remedy to the large memory requirements are to choose some of the primitives to be not only elementary operations, but larger pieces of code. In this case customized primitives need to be constructed. This method has the advantage of keeping the computational cost low, contrary to checkpointing which might be inefficient. In general one needs to decide on a balance between memory cost and computational complexity for optimal performance.

Extension to complex variables

So far we introduced automatic differentiation for real functions of real variables. Now we will deal with the problem where the input vectors have complex elements and therefore $\boldsymbol{f} : \mathbb{C}^{m_0} \to \mathbb{R}^{m_N}$ is a real differentiable function of complex variables. The output space is always real when AD is used to optimize some function, which is the case of interest for this thesis. Real functions of complex variables can be shown to be either constant or not complex differentiable using the Cauchy-Riemann equations, and therefore one needs to be more careful on how to define the corresponding forward and reverse procedures. Partial derivatives can still be defined, namely the Wirtinger derivatives [Wir27]:

$$\begin{aligned} \frac{\partial}{\partial z_i} &:= \frac{1}{2}\left(\frac{\partial}{\partial x_i} - i\frac{\partial}{\partial y_i}\right) \\ \frac{\partial}{\partial z_i^*} &:= \frac{1}{2}\left(\frac{\partial}{\partial x_i} + i\frac{\partial}{\partial y_i}\right) \end{aligned} \tag{1.11}$$

With these definitions, when differentiating with respect to z, z^* is treated as a constant, while the z is considered constant when differentiating with respect to z^*.

From these we can define the nabla operator [Spe64]:

$$\boldsymbol{\nabla} = 2\frac{\partial}{\partial \boldsymbol{z^*}} = 2\left(\frac{\partial}{\partial z_1^*}, \frac{\partial}{\partial z_2^*}, \cdots, \frac{\partial}{\partial z_{m_0}^*}\right), \tag{1.12}$$

Since $\boldsymbol{f}$ maps to $\mathbb{R}^{m_N}$, the gradient of its $k-$th element relates to the Wirtinger derivatives through the relation:

$$\boldsymbol{\nabla} f_k(\boldsymbol{z}) = 2\frac{\partial f_k}{\partial \boldsymbol{z^*}} = 2\left(\frac{\partial f_k}{\partial \boldsymbol{z}}\right)^* \tag{1.13}$$

which is consistent with the equivalent definition for real variables.

1.2 Basics on tensor network states

1.2.1 Physical motivation

Quantum many-body systems are often studied through effective low-energy descriptions that simplify their analysis. When it comes to simulating strongly interacting systems, it

is typically useful to discretize the full Hilbert space in the basis of real coordinates, thus obtaining a lattice upon which we can define the effective Hamiltonian. At each site i of the lattice, a local basis $|a_i\rangle$ is defined according to the local degrees of freedom (e.g. spin, occupation number, etc.). A general eigenstate for a Hamiltonian on a lattice of N sites and $d-$dimensional local Hilbert space takes the form:

$$|\psi\rangle = \sum_{a_i=1}^{d} C_{a_1 \dots a_N} |a_1 \dots a_N\rangle \tag{1.14}$$

Performing efficient computations with a random state drawn from such an ansatz in the Hilbert space $\mathcal{H} = \left(\mathbb{C}^d\right)^{\otimes N}$ is computationally very inefficient for classical computing methods. Classical computers are, however, very successful in performing calculations efficiently with many-body states of low entanglement. Physically, the difference is that two subsystems of a random state in $\mathcal{H}$ are most probably entangled with each other no matter their position. However, physical interactions are usually of finite range, i.e. local interactions described by Hamiltonians of the form:

$$H = \sum_i \hat{h}_{i,n_i} \tag{1.15}$$

where n_i are a finite set of sites with which site i directly interacts. For such Hamiltonians, strong correlations tend to be shared only among the sites that directly interact with each other. In particular, the entanglement entropy of the ground states of local Hamiltonians that have a finite energy gap between ground and excited states satisfies an *area law:* if we cut a partition R from the ground state and measure the entanglement entropy $S(\rho_R)$ of the corresponding reduced density matrix ρ_R of R, the entanglement grows with the number of neighbors of the sites that live in the boundary of region R, i.e. with the boundary surface area of R (see Fig. 1.11a):

$$S(\rho_R) \sim \partial R \tag{1.16}$$

The reason why neighboring sites tend to be strongly entangled in the first place has to do with the fact that, usually, for local Hamiltonians, it is energetically efficient for neighboring sites to be as much entangled as possible, only limited by entanglement monogamy [CPGSV21]. Eq. (1.16) have been proved for one-dimensional local gapped Hamiltonians [Has07]. For higher dimensions there is no general proof, but several area laws exist [PEDC05] [CE06]. Critical systems might need corrections with respect to area laws [GK06], [Wol06], [CEP07].

Tensor network states offer efficient representations of states in $\mathcal{H}$ with entanglement structure motivated by such physical considerations. A tensor network state is a contraction of tensors which forms tensor C corresponding to state (1.14) (or to the density matrix for mixed states, which we are not interested in in this thesis). Tensor network states become useful when a (in general) high-degree tensor C can be decomposed to tensors of small degrees. This is the case for ground states of physical systems, which makes tensor networks extremely useful for such problems. Different families of tensor network states correspond

to differently distributed entanglement across the lattice, and therefore the geometry of the lattice has to be chosen so that it captures the physics related to the system's correlations in an efficient way. Finally, even though often tensor networks represent quantum states, they can also be used to represent other objects such as expectation values, classical partition functions, operators, and more.

1.2.2 Diagrammatic notation

It is very practical and instructive to use diagrams to describe tensor networks and tensorial operations. In the context of this thesis, tensors are considered simply multidimensional arrays and are denoted by a shape (circle, square, triangle, etc.), together with lines which correspond to each dimension. The number of dimensions of the array is called *degree*. For example a degree-three tensor $T \in \mathbb{K}^{d_a} \times \mathbb{K}^{d_b} \times \mathbb{K}^{d_c}$ ($\mathbb{K} = \mathbb{R}$ or $\mathbb{C}$) is given by the diagram:

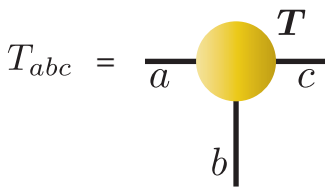


Figure 1.4: Example of degree-three tensor.

where $a, b, c \in \{1, ..., d_{a,b,c}\}$.

The ordering of the legs of the tensor is in general important. Tensor contractions are depicted by connecting the lines at the contracted dimensions. For example, the contraction of the previously defined tensor T at the third index, with $S \in \mathbb{K}^{d_c} \times \mathbb{K}^{d_d}$, is given by the shape:

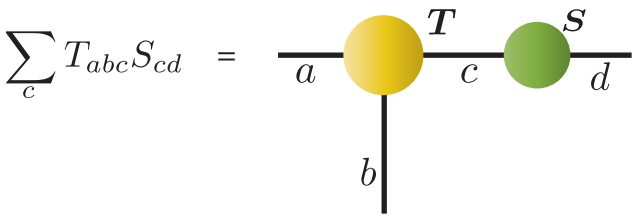


Figure 1.5: Example of contraction of two tensors.

Gradients of tensors can be represented diagrammatically simply by removing the corresponding tensor from the diagram. For example, given the contraction $X_{abd} = \sum_c T_{abc} S_{cd}$ of Fig. (1.5), the gradient $\frac{\partial X_{abd}}{\partial T}$ is represented as:

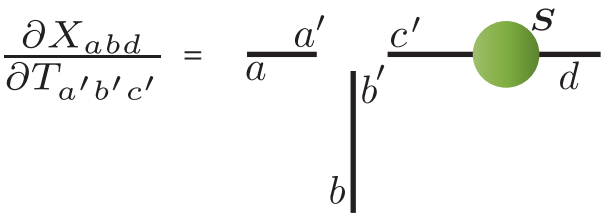


Figure 1.6: Example of tensor derivative. Lines without a specified tensor contracted to them represent identity matrices.

An important property of tensor networks is *gauge freedom*. Namely, to any contracted bond, an invertible matrix can be applied together with its inverse without changing the tensor network (see Fig. 1.7). This means that there is no unique representation for tensor networks.

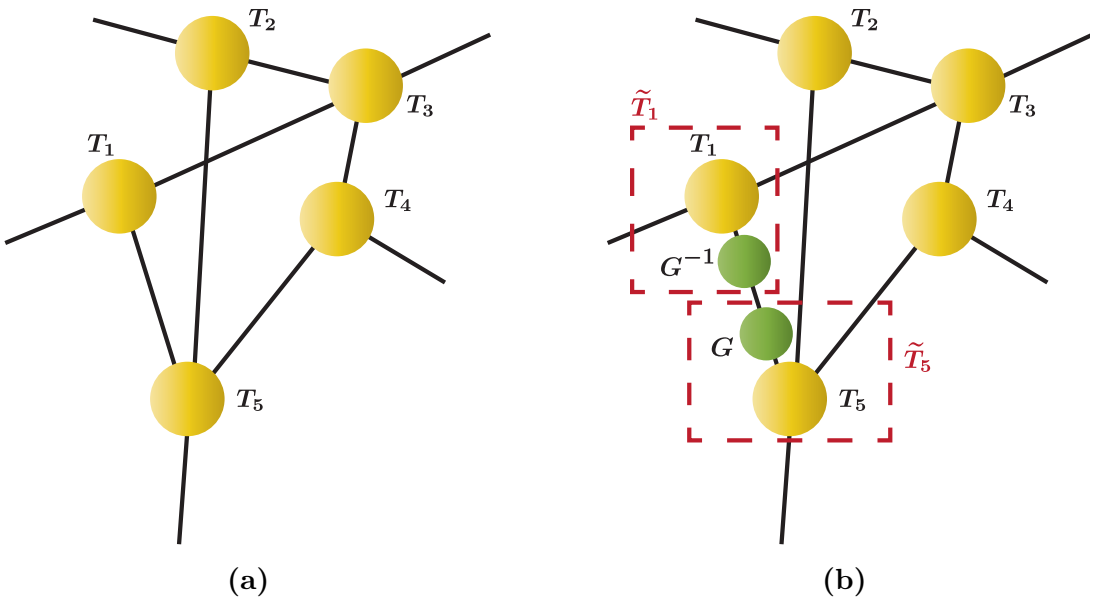


Figure 1.7: (a) Arbitrary tensor network state made up of 5 tensors. (b) Introducing a gauge G to bond, for example, 1-5, and defining $\tilde{T}_1 = T_1 G^{-1}$ and $\tilde{T}_5 = G T_5$, leaves the total network invariant.

1.2.3 Implementation of symmetries

Quantum many-body systems are often governed by Hamiltonians with a number of discrete or continuous symmetries. Moreover, distinct phases might be characterized by ground states invariant under different symmetries. In computational physics, the existence of symmetries should be taken into account when possible as it can significantly speed up the calculations as well as achieve more accurate results. The reason is that when an array

is invariant under some operation, it can be decomposed as the direct sum of different blocks of smaller arrays. Because of this, multiplications of large arrays can be replaced by multiplications of smaller blocks of arrays, thus reducing the computational costs.

Symmetries can be incorporated in the representation of tensor networks via extra labels in the indices which specify the symmetry sectors. For example, consider a U(1) symmetry connected to a conservation of charge. A degree-three tensor is now represented (in the charge-operator eigenbasis) as:

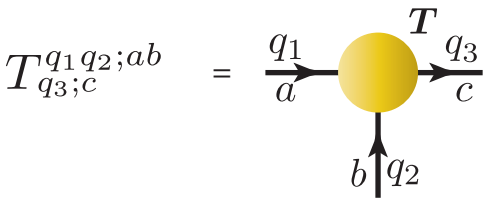


Figure 1.8: Example of tensor subscribing to a U(1) symmetry.

with the charge quantum numbers q_1, q_2, q_3 being restricted by a conservation law, $q_1+q_2 = q_3$ in accordance with the U(1) invariance. The conservation law is encoded diagrammatically through arrows on the legs of the tensor according to the flow of charge. In written form, the convention is to set up/down indices for incoming/outgoing charges. In this case, the labels a, b, c label the states in a specific symmetry sector of charge $q_{1,2,3}$. Another example used for the benchmarks of this thesis is the SU(2) symmetry. In this case, an extra label $q_{i,z}$ is added to distinguish individual states of a multiplet.

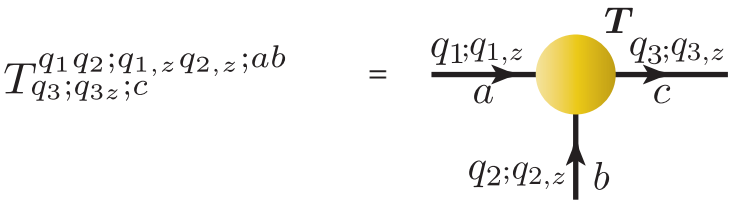


Figure 1.9: Example of tensor subscribing to a SU(2) symmetry.

This is a very powerful technique to automatize symmetry tracking in tensor networks. For this reason, the use of the QSpace library was chosen, which keeps track of the symmetry labels and conservation laws. The latest documentation for QSpace is reported in [Wei24].

1.2.4 The (infinite) Projected Entangled Pair States

Among all the possible tensor network geometries, the Projected Entangled Pair States (PEPS) are those that efficiently capture the entanglement properties of ground states of local, gapped Hamiltonians. They were originally proposed as a generalization of the Matrix Product States (MPS) for 2D systems [VC04]. Each tensor of a PEPS corresponds to a quantum system (spin, boson, fermion, etc.) and consists of one physical index to which are assigned the quantum numbers related to that system; the correlations between

particles are encoded via z virtual indices, where z is the coordination number of that site. The virtual bonds in a tensor network are those that do not correspond to physical degrees of freedom; however, they do have a physical significance as they are linked to how entanglement gets distributed through the system's interactions. In this thesis, square-lattice geometries are studied, and therefore we are interested in systems in 2D where the individual tensors have the shape:

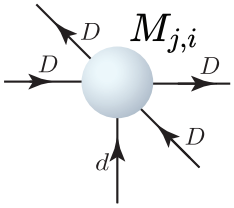


Figure 1.10: Elementary PEPS tensors of a 2D PEPS arranged in a square geometry. The letters next to the arrows indicate the dimension of the space of the corresponding index. The virtual indices have a dimension D, while the local index dimension d.

The whole tensor network can be visualized as in Fig. 1.11b.

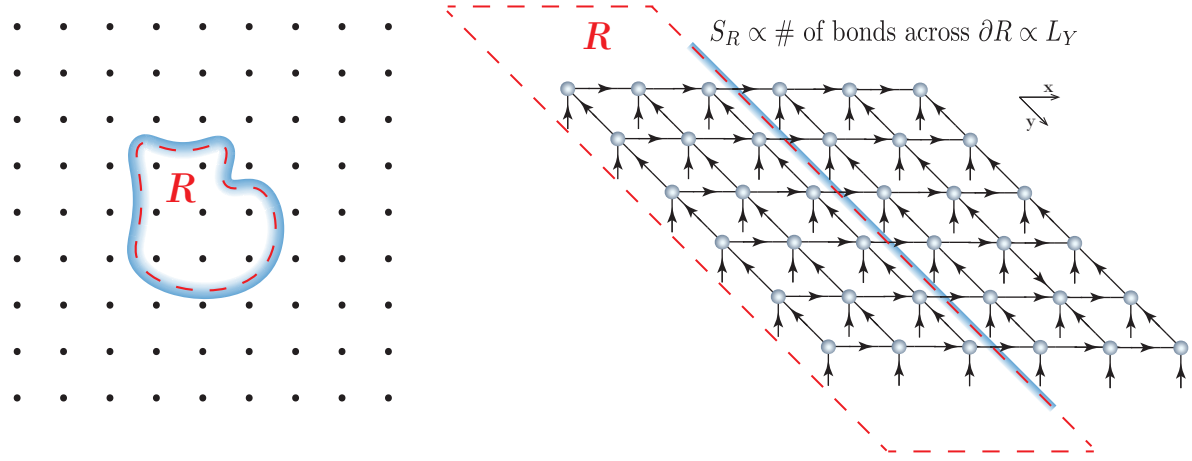
The PEPS geometry might seem intuitively the natural choice for systems where the leading interaction is between nearest neighbors, however the ansatz is far more general: any state of the form (1.14) with finite d can be represented through PEPS [ECP10]. Specifically, the maximal entanglement entropy for a PEPS with virtual bond dimension D grows as:

$$S_{R,max} \sim \partial R \log D \tag{1.17}$$

which means that the smaller the bond dimension D the smaller the amount of entanglement that the PEPS can capture. However, for ground states governed by the area law for entanglement (1.16), D can take small values and still permit very accurate calculations. What is important is that the area law holds true for any bipartition, which is indeed the case. An example of this is illustrated in Fig. 1.11b. If we want to represent a highly entangled state, then the bond dimension must be increased exponentially with system size, which means that this representation is not efficient for such states.

The thermodynamic limit can be approached with PEPS by considering a *supercell* of tensors which is repeated across the network. The size of the supercell determines the exact way in which translational symmetry presents itself on the lattice. Such a construction is abbreviated as *iPEPS*, originally introduced in [JOV+08]. For an $L_X \times L_Y$ supercell the degrees of freedom grow linearly with the size of the supercell as $dD^4 L_X L_Y$, and stay constant with increasing system size, keeping the iPEPS representation efficient.

So far we have not specified whether the particles that make up the quantum state are bosons or fermions. For bosonic statistics, no modifications to the PEPS ansatz are required since the creation operators commute. However, for building a PEPS that can capture ground-state properties of local fermionic Hamiltonians, two essential modifications are required [COBV10]:



(a) (b)

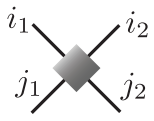
Figure 1.11: (a) Lattice of particles. The entanglement of the particles inside area R, with the rest of the lattice, grows with ∂R (blue shaded area). (b) Diagrammatic representation of PEPS. The PEPS construction automatically satisfies the area law. The entanglement of the particles in bipartition R with the rest increases only with the number of bonds at the boundary, and thus, in this example, with L_Y. The same holds true for all bipartitions.

1. Tensors M should be invariant with respect to the parity of the fermionic particle number. This comes from the fact that fermionic Hamiltonians possess this $\mathbb{Z}_2$ symmetry, and therefore their eigenstates should correspond to fixed parity. For this, the elements that mix parities must be set to 0:

$$M^{a\ c\ e}_{\ b\ d} = 0 \qquad \text{if} \quad p(a)p(b)p(c)p(d)p(e) = -1 \tag{1.18}$$

   where $p(x)$ is the parity carried by the edge x.

2. At each line crossing in the tensor network, a fermionic SWAP operator should be added, defined as:



$$= \delta_{i_1 j_2}\delta_{i_2 j_1} S(i_1, i_2)$$

   with

$$S(i_1, i_2) = \begin{cases} -1 & \text{if } p(i_1) = p(i_2) = -1 \\ 1 & \text{otherwise} \end{cases} \tag{1.19}$$

   This comes from the fact that such line crossing represents exchange of the degrees of freedom carried by the crossed edges [CV09] (be it virtual or not). The fermionic

SWAP assures that the fermionic statistics will be satisfied since a minus sign is introduced for each exchange of two fermions. For what it means for bonds to carry fermionic charge, see 1.2.3.

These simple modifications to the original PEPS definition result in an efficient way of studying fermionic ground states. By the second rule, the new PEPS representation has the diagrammatic form of Fig. 1.12.

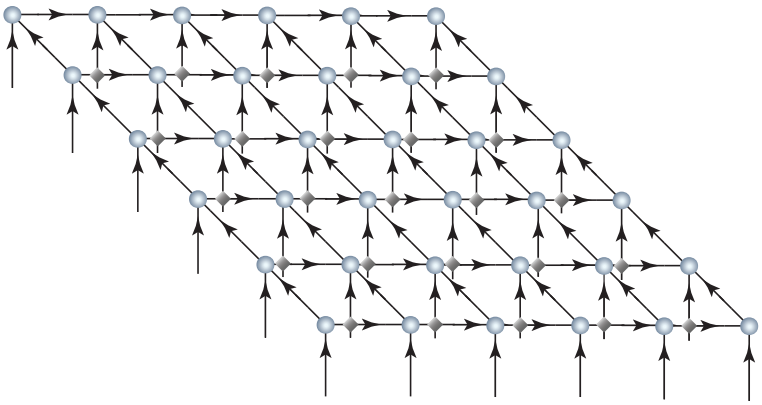


Figure 1.12: Generalization of the PEPS of Fig. 1.11b to fermions. Fermionic SWAPs are added whenever two lines cross.

With such a PEPS, expectation values can be computed by defining the conjugate to the PEPS and the corresponding *double-layer tensors*. For each tensor $M_{i,j}$ of the supercell the corresponding double-layer tensor $\mathcal{M}_{i,j}$ is defined from the contraction of the physical index of $M_{i,j}$ and its conjugate $W_{i,j}$, as shown in Fig. 1.13.

1.3 AD for the Corner Transfer Matrix Renormalization Group

In this section the application of AD (Sec. 1.1) to tensor networks is exemplified for the case of the Corner Transfer Matrix Renormalization Group (CTMRG). CTMRG is one of the most competitive algorithms for performing calculations with ground states of 2-D lattices at the thermodynamic limit, and for this reason it has been chosen to compute the energies of iPEPS for the benchmarks of Ch. 3. I will start by introducing the CTMRG algorithm, and subsequently continue with details on how to implement backpropagation efficiently with it. Since AD is a generic method for gradient calculation, it can be applied to all kinds of different algorithms; therefore, one can apply these ideas to other cases.

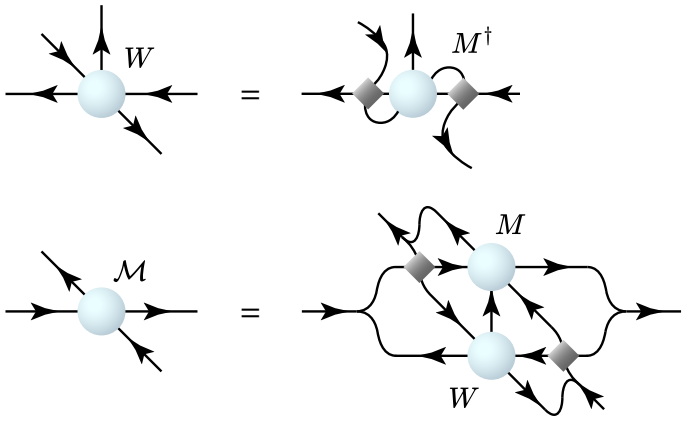


Figure 1.13: Conjugate W and double-layer tensor $\mathcal{M}$ with respect to the M tensors that make up the iPEPS.

1.3.1 Corner Transfer Matrix: Forward sweep

A most challenging aspect of tensor network states drawn from an iPEPS ansatz, or even from finite PEPS, is computing expectation values. The reason behind it is the lack of a canonical form similar to tree-like structures[2]. As explained in App. A, the canonical form allows the use of isometric relations to efficiently contract the environment of a tensor of a MPS, and can also be used to approximate infinite environments. The lack of a canonical form is the biggest bottleneck for iPEPS, and different methods for contracting the infinite environment must be constructed. For this purpose, several approximate methods have been developed, among which is the Corner Transfer Matrix (CTM), originally adapted in the context of iPEPS in [OV09].

Below, CTMRG is described following the implementation in [Zha21]. For more details, this source should be used along with the references therein. Fig. 1.13 and Fig. 1.15-1.19 explaining the steps of CTMRG were kindly provided by Changkai Zhang.

Suppose that we are equipped with an approximation of the ground state of a Hamiltonian, drawn from the iPEPS ansatz for some bond dimension D. The CTM ansatz encodes the infinite environment of each tensor in the supercell in 8 tensors called *corner matrices* and *transfer matrices*. The mapping of iPEPS to CTM is shown in Fig. 1.14 for a central tensor $\mathcal{M}_{1,1}$.

[2]An *isometric* PEPS ansatz was defined in [ZP20], which, however, does not represent a generic PEPS ansatz.

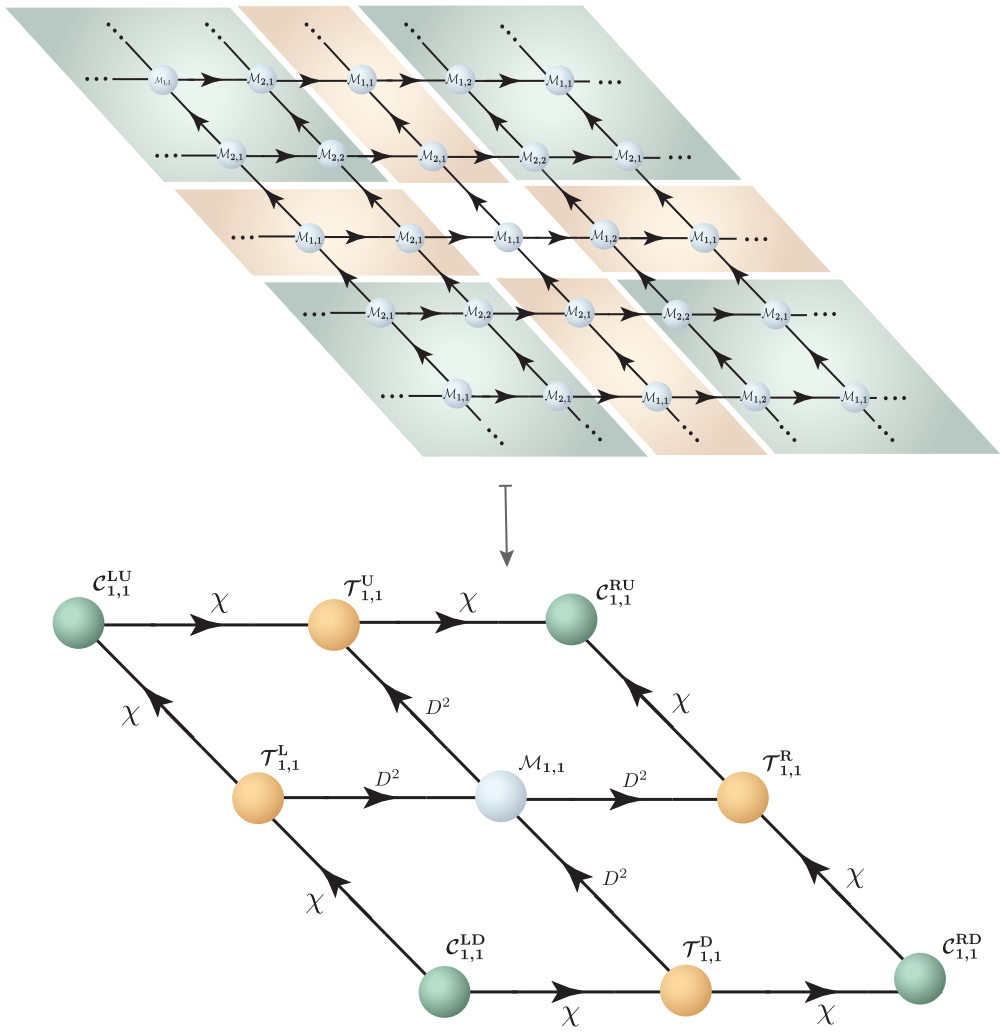


Figure 1.14: Mapping of the iPEPS ansatz (top) to the CTM ansatz (bottom) for tensor $\mathcal{M}_{1,1}$. The shaded areas of the top diagram are encoded to the similarly colored tensors of the bottom diagram. The value χ is the environment bond dimension and its exact value is case-dependent.

This mapping is achieved via an iterative coarse-graining procedure, the CTMRG. CTMRG consists of the following steps:

- **Initialization:** Choose an initialization of the corner and transfer matrices. A natural choice is initializing with the immediate environment of each tensor as shown in Fig. 1.15.

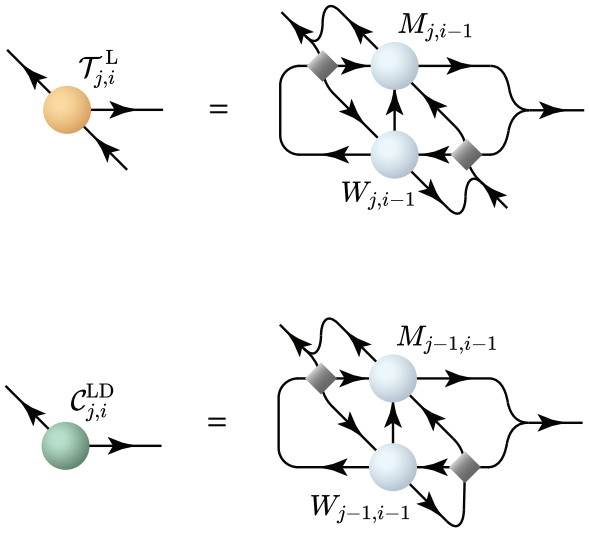


Figure 1.15: Exemplary initialization of the corner matrices $\mathcal{C}^{LD}_{j,i}$ and transfer matrices $\mathcal{T}^{L}_{j,i}$ using the closest tensors to the site (j,i). Similar initialization can be defined for the rest of the environment tensors.

- **Vertical update:** We start with vertically updating the environment. This consists of three steps:

  1. Renormalization Group (RG): The goal of this step is to obtain an efficient description of the environment tensors, i.e. obtain environment tensors whose individual dimensions do not exceed the pre-selected value χ. This is achieved via subsequent Singular Value Decompositions (SVD) and truncation of the smallest singular values, as explained in detail in Fig. 1.16. As the coarse-graining proceeds, the environment bond dimension grows by D^2 at each iteration, and therefore this step is crucial. At this step projectors with dimensions at the most $\chi D^2 \times \chi D^2 \times \chi$ are obtained.
  2. Update corner matrices: A transfer matrix is absorbed by a corner matrix to obtain the new corner, as shown in Fig. 1.17. This contraction increases the bond dimension of the corner matrices from $\chi \times \chi$ to $\chi D^2 \times \chi$. Then, by contracting with the projectors computed in the first step the dimension decreases back to $\chi \times \chi$.
  3. Update left/right transfer matrices: Similarly to the corner matrices, the transfer matrices are updated by absorption of a double-layer $\mathcal{M}$ tensor, as shown in Fig. 1.16. Again, renormalization is required to keep the environment bond dimension no larger than χ.

- **Horizontal update**: Similar to the vertical update, but now the renormalization involves SVD on the left-right splitting of the iPEPS, resulting in 4 new projectors. The corner matrices are updated horizontally and the up/down transfer tensors are now updated.

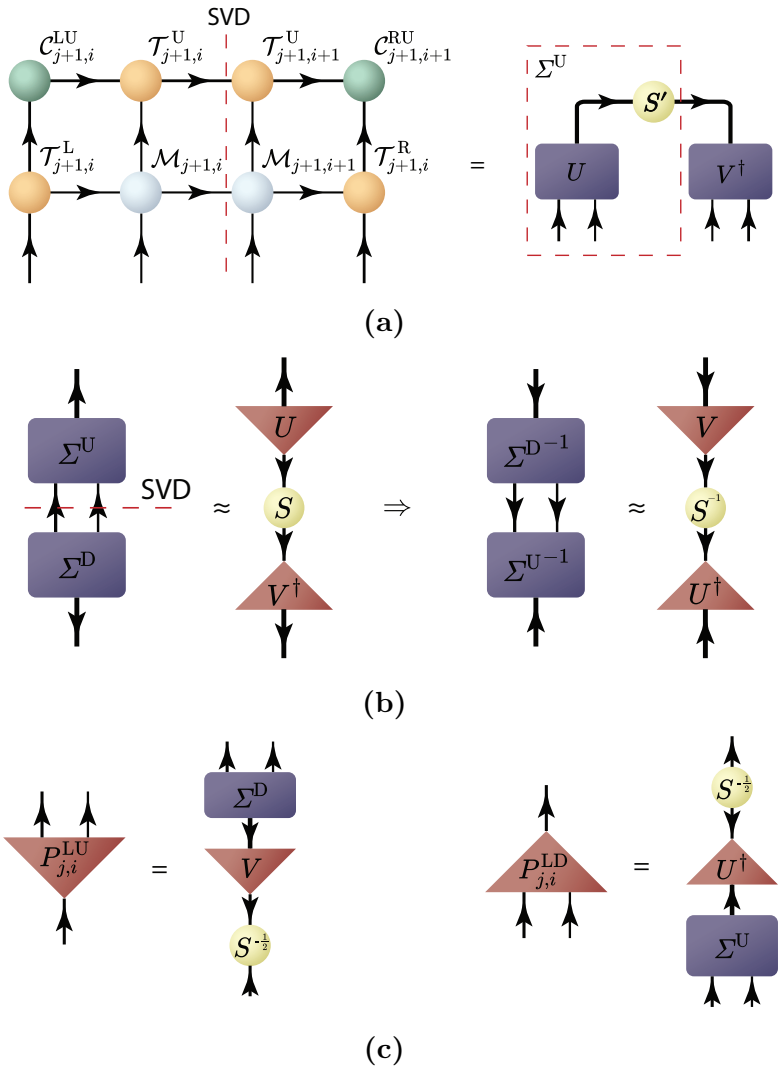


Figure 1.16: Update of up/down projectors for the renormalization. (a) Firstly, an effective description of the upper part of the iPEPS is computed by contractions that form the left part of Fig. (a). Subsequently, a SVD and truncation of the smallest singular values assures stability of the renormalization step. In this way, the tensor Σ^{U} of Fig. (a) is obtained as an efficient description of the upper left part of the iPEPS. Similarly for the lower part of the iPEPS tensor Σ^{D} is obtained. (b) $\Sigma^{U}\Sigma^{D}$ is computed and after an SVD the inverses of the three SVD tensors are taken. At the SVD, up to χ singular values are kept. (c) From there two projectors can be defined, P^{LU} and P^{LD}, which map from a space of dimension χD^2 of the left upper/lower iPEPS to one of dimension χ. With this definition, the product $P^{\mathrm{LU}}P^{\mathrm{LD}}$ is the identity.

- Steps 1-3 are repeated for vertical and horizontal updates alternately, until convergence.

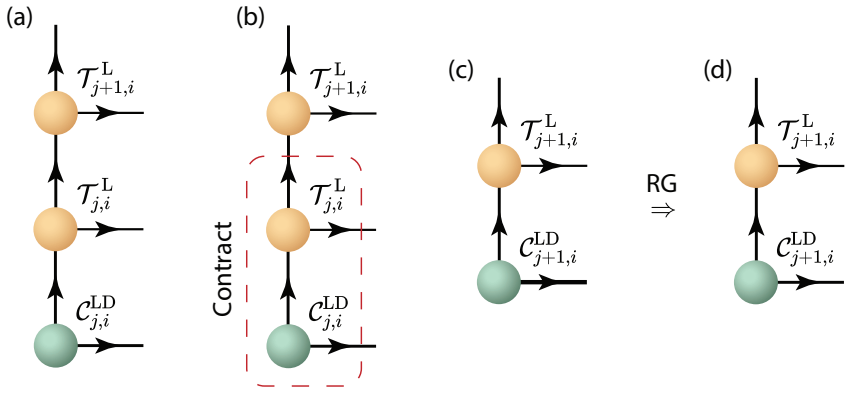


Figure 1.17: Left-down corner matrix update of CTM. (a) Setup from previous iteration. (b) Absorbing a transfer matrix to the corner matrix. (c) Updated corner matrix, with the increased bond dimension χD^2 shown with a thicker line. (d) Renormalization decreases this dimension back to χ.

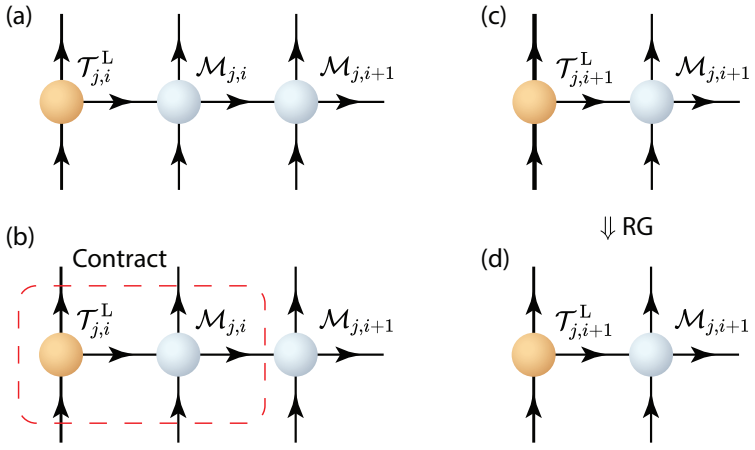


Figure 1.18: Left transfer matrix update of CTM. (a) Setup from previous iteration. (b) Absorbing a double-layer tensor to the transfer matrix. (c) Updated transfer matrix with the increased bond dimensions shown with thicker lines. (d) Renormalization.

The forward sweep starts with initializations 1.13 and includes all the subsequent steps of CTMRG. Finally, it concludes with the computation of the expectation value of the Hamiltonian. In this thesis, nearest and next-nearest neighbor interactions are studied. The nearest-neighbor terms of the Hamiltonian for the square iPEPS act on vertically and horizontally neighboring sites. The next-nearest neighbor terms act on the diagonally and anti-diagonally neighboring sites. The diagrammatic form of such expectation values is shown in Fig. 1.19.

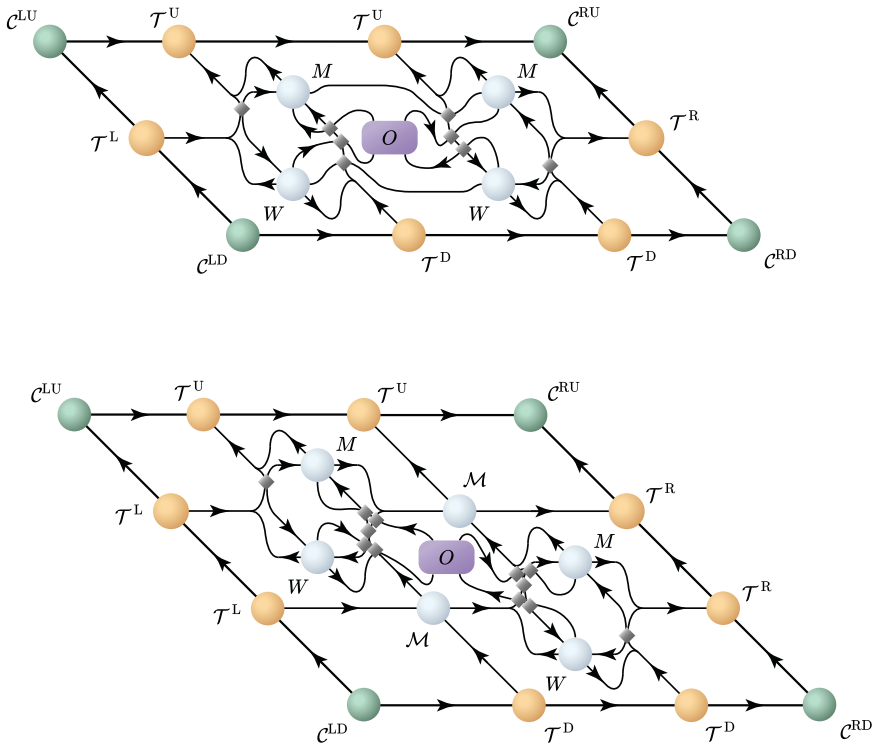


Figure 1.19: Top: Expectation value of horizontal two-site observable. Bottom: Expectation value of anti-diagonal two-site observable.

1.3.2 Corner Transfer Matrix: Reverse sweep

To conduct gradient-based ground-state search, what we are interested in is exploiting AD for calculating energy gradients. The function f through which we would like to back-propagate and compute its gradient is only a function of the tensors $\{M_{i,j}\}$. Summarizing the forward sweep, f consists of:

- Initializations 1.13 for all the $L_X L_Y$ tensors of the supercell
- Initializations 1.15 for all the $8L_X L_Y$ environment tensors
- CTMRG for all the $8L_X L_Y$ environment tensors vertically and horizontally updated, repeated N_{it} times until convergence

- The expectation value of the Hamiltonian for vertical and horizontal interaction terms (and across the two diagonals for the next-nearest neighbor interaction) for all $L_X L_Y$ tensors (Fig. 1.19). The total energy E is calculated as the average over all these terms.

What we wish to compute are the gradients:

$$\overline{E}_{i,j} = \left.\frac{\partial E}{\partial M_{i,j}}\right|_{M_{0i,j}} \tag{1.20}$$

using reverse-mode AD, starting from an initial iPEPS ansatz with parameters $\{M_{0i,j}\}$. To do this, the whole computational graph of all the above steps needs to be recorded and stored in memory at the forward sweep. Operations $\boldsymbol{\phi^{(i)}}$ which map from one node of the graph to another include contractions, SVDs, permutations, etc. This information on how the different nodes are connected also needs to be stored in memory, so that the corresponding gradient formula can be addressed at the backward pass. A few non-trivial gradient formulas for such common tensor network operations are given in the app. B. At first sight, calculating Eq. (1.20) in this way seems an unfeasible task, as there is a huge number of contractions and intermediate tensors which need to be stored in memory. In particular, for our benchmarks it was found that already at bond dimension $D = 5$ the consumption in memory was more than 20GB, and with increasing bond dimension it quickly reached hundreds of GB, since the memory cost of CTMRG scales as $\mathcal{O}(D^4\chi^2)$. The challenging part is the fact that, within the forward pass, there exists an iterative procedure which increases the memory costs proportionally to the number of iterations and supercell-size. In order for AD to become practically feasible for the CTMRG coarse graining, we perform the checkpointing technique explained in Sec. 1.1. As an example, Fig. 1.20 explains how updating a corner matrix and its gradients can be done via checkpointing. Another possible way to tackle the problem is the backward formula for fixed-point iterations [Chr94], which is briefly described in App. B.

A reasonable choice of checkpoints to the RG step leads to the computational graph of Fig. 1.21. In general, the checkpoints should be customized depending on the computational and memory requirements for each algorithm; however, a balance between the memory and computational cost should be preferred. An example where the computational burden of checkpointing outweighs its benefits is for the calculation of vector-Jacobian products for the corners of the iPEPS, made up of the tensors $\{\mathcal{C}, \mathcal{T}^{U/D}, \mathcal{T}^{R/L}, M, W\}$. These corners are needed for the update of the projectors (see Fig. 1.21). Even though checkpointing leaves the computational complexity unaltered, it becomes impractical, since for each iteration it includes contractions with scaling $4\mathcal{O}(\chi^3 D^4)$ (and $6\mathcal{O}(\chi^3 D^4)$ for fermionic iPEPS), instead of $\mathcal{O}(\chi^3 D^4)$. Avoiding the checkpointing for this part comes at a cost of storing four times more tensors (and six for fermionic iPEPS). As we can see from this example, checkpoints should be applied with care. Except for the RG step, checkpoints should also be applied for the calculation of the energy, as it also involves a big number of intermediate, large tensors.

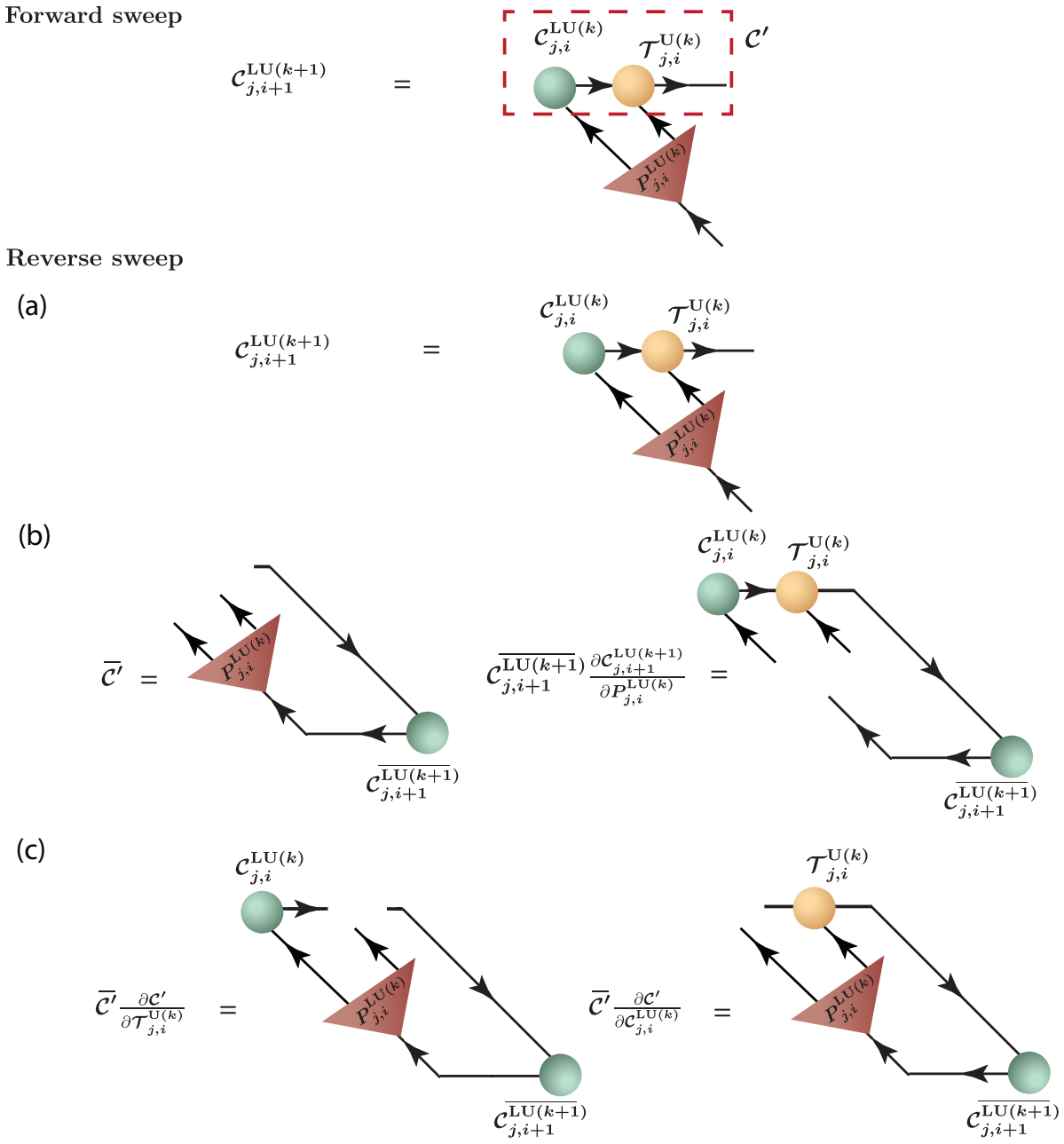


Figure 1.20: Forward and reverse sweeps of the update of a corner matrix at CTMRG iteration $k+1$. At the forward sweep, the initial and final tensors are saved to memory while the intermediate tensor $\mathcal{C}'$ is not. At the reverse sweep, first the forward calculation is re-executed (step (a)), this time saving temporarily also $\mathcal{C}'$. Gradient $\mathcal{C}_{i,j}^{\overline{\mathrm{LU}(k+1)}}$ is known from previous steps, and can be used to compute the gradients with respect to tensors directly related to it (step (b)). Lastly, the two remaining gradients are computed (step (c)). Once all the gradients have been updated, unnecessary information of doubly computed the tensors can be discarded.

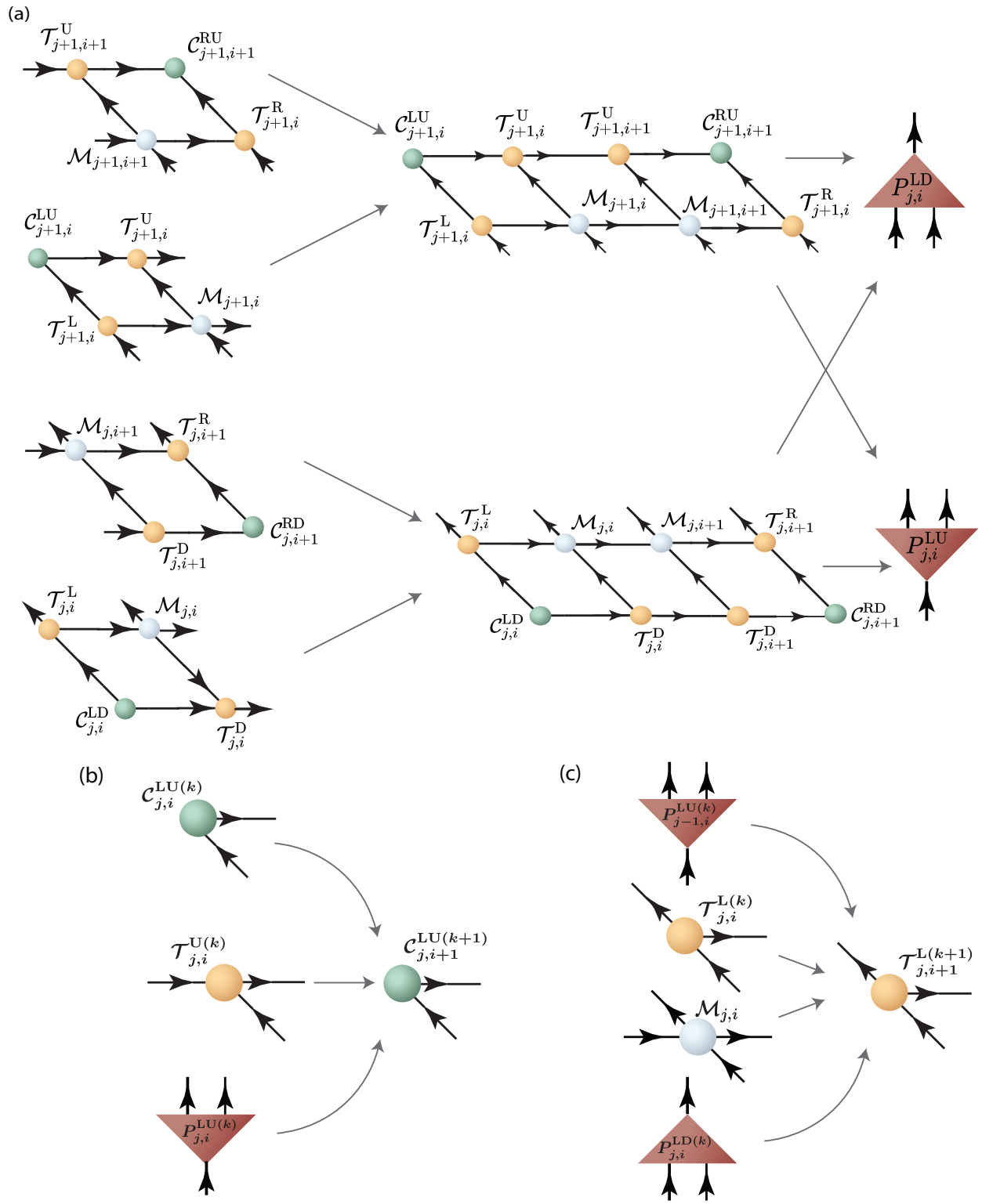


Figure 1.21: Computational graph for the RG step of CTMRG. (a) Renormalization step. (b) Update of corner tensors. (c) Update of transfer tensors.

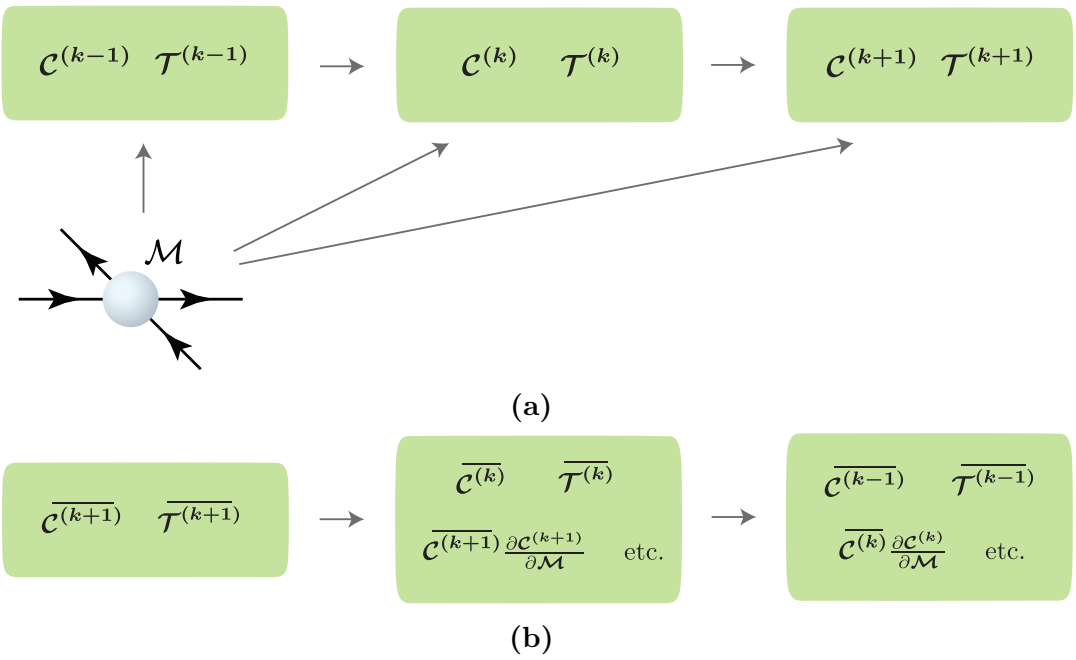


Figure 1.22: (a) Coarse-grained schematic of the computation graph of CTMRG. Environment tensors of iteration $(k-1)$ depend directly only on those of iteration k. (b) Gradient calculation can be done iteration-by-iteration at the reverse sweep, so that the part of the graph related to iteration $k+1$ can be discarded once the the gradients of iteration k are obtained.

In parallel, during the backward pass, storage space should be available for the gradient tensors. The gradient tensors should also be saved, as they contain useful information which might be reused in different steps of the backward process. For example, looking at backpropagating through the graph (b) of Fig. 1.21, $\overline{\mathcal{C}^{\mathrm{LU}(k+1)}_{j,i+1}}$ is computed once and saved in order to be used for all three vector-Jacobian products of its parent nodes (see steps (b)-(c) of Fig. 1.20). In order to avoid unnecessary memory consumption because of this process, a simple trick is to use that in a coarse-grained schematic of the computational graph (Fig. 1.22a), the environment tensors of iteration $k-1$ do not depend directly on those of iteration $k+1$. Therefore, if one computes all the gradients up to iteration k, all the information regarding the computational graph and the gradients from iteration $k+1$ and onward can be discarded, making space for next gradient computations. This trick is useful for any algorithm dependent on iterative procedures similar to CTMRG, and it only comes down to rearranging the order in which the gradient computations are made (see Fig. 1.22b).

Using all of the above, we can estimate that this implementation of reverse-mode AD for CTMRG requires $40 L_X L_Y N_{it}$ saved tensors (or $56 L_X L_Y N_{it}$ for fermionic iPEPS). Even though this is still a big number, the order of magnitude of total memory consumption

required for the benchmarks of Ch. 3 was still reduced from hundreds of GB with plane AD, to tens of GB after applying the above tricks, at the expense of increasing the computational cost to about twice as much.

Finally, some comments are required regarding the stability of the backward process. Specifically, the implementation of the reverse AD for the SVD might create instability to the gradient and should be applied as presented in App. B. Additionally, due to the terms S^{-1} containing inverse singular values at the update of the projectors (see Fig. 1.16), the gradient of such terms $\sim S^{-2}$ is prone to cause instabilities at the backward pass of the CTMRG. For this reason, it is advisable to keep singular values no smaller than 10^{-8} at the corresponding SVD (Fig. 1.16b). This introduces a truncation error to the energy calculations (which is already partly there since only the χ largest singular values are kept).

Chapter 2

Gradient-based optimization of tensor networks

In quantum mechanics, variational methods are the most powerful technique for ground-state approximation when perturbative expansions cannot be used. The variational approach assumes that a suitable variational ansatz is known for the ground state, and one needs to compute the variational parameters of the ansatz so that the corresponding state belongs to the subspace of the lowest eigenvalue of the Hamiltonian. In the context of this thesis, the variational ansatz is the iPEPS which takes as parameters the (single-layer) tensors $\{M_{i,j}\}$ of the supercell. $\{M_{i,j}\}$ need to be optimized to approximate as much as possible a true ground state. Optimization methods are guaranteed to give an upper bound to the ground-state energy due to the variational principle. Given an iPEPS ansatz $|\psi\rangle = |\psi(\{M_{i,j}\})\rangle$, the variational principle states that:

$$E_0 \leq \min_{\{M_{i,j}\}} \frac{\langle\psi|H|\psi\rangle}{\langle\psi|\psi\rangle} \tag{2.1}$$

where H is the full Hamiltonian and E_0 the true ground-state energy. Once we are employed with an algorithm to calculate the expectation value of the Hamiltonian, optimization algorithms can be used to compute, or at least approximate, the right-hand side of Eq. (2.1).

For this purpose, gradient-based optimization methods can be employed, where gradient information is used to approach the ground state by minimizing the energy. AD as described in Ch. 1 provides us an efficient method to calculate energy-gradient values. There are several gradient-based optimization techniques which can be used for tensor networks. The most naive approach would be gradient descent, where the update rule:

$$\begin{gathered} \boldsymbol{x} \to \boldsymbol{x} - \eta \nabla E(\boldsymbol{x}) \\ \text{with } E(\boldsymbol{x}) = \frac{\langle\psi(\boldsymbol{x})|H|\psi(\boldsymbol{x})\rangle}{\langle\psi(\boldsymbol{x})|\psi(\boldsymbol{x})\rangle} \text{ and } \eta > 0 \end{gathered} \tag{2.2}$$

is used for the variational parameters $\boldsymbol{x}$, which are vectors containing the elements of the iPEPS tensors. However, such a method might take a very long time to converge when

the number of parameters is large. The exact rate of convergence depends on the objective function, but in general it can be linear or slower [NW99]. For this reason, other methods which converge quadratically have been developed and are used for tensor networks, such as the conjugate gradient method [FR64] and Newton-type methods [NW99]. For the benchmarks of this thesis the limited-memory BFGS method was used, which is in the class of quasi-Newton methods. In Sec. 2.1 this method is described, along with some necessary background. In Sec. 2.2 possible initializations of the tensor network are given, along with some details which can improve the performance or efficiency of the optimization.

2.1 Quasi-Newton optimization techniques

2.1.1 The BFGS optimization

Suppose that we have a function $f : \mathbb{R}^{m_0} \to \mathbb{R}$ twice continuously differentiable and we want to find a global minimum $\boldsymbol{x} = \boldsymbol{x_e}$. For non-convex functions this is a very difficult task because it requires knowledge of the function and its derivatives for all $\boldsymbol{x} \in \mathbb{R}^{m_0}$. Optimization algorithms are in general capable of scanning a restricted region of the function and find a local minimum. The BFGS optimization method, named after its inventors Broyden, Fletcher, Goldfarb, and Shanno, is a powerful technique that combines numerical accuracy and efficiency in search for (local) minima. It belongs to the class of *line search* methods, which means that it is composed of successive iterations $k = 0, 1, ...$ in which the variables are updated as:

$$\boldsymbol{x_k} \to \boldsymbol{x_{k+1}} = \boldsymbol{x_k} + a_k \boldsymbol{p_k} \tag{2.3}$$

Here $\boldsymbol{x_k}$ is the variable value at the k-th step of the optimization process, $\boldsymbol{p_k}$ is the *search direction*, a direction along which the function is guaranteed to decrease, and $a_k > 0$ is the *step length* calculated through a line search algorithm for each iteration k. a_k should be chosen so that the function value minimizes in a given search interval. Line search methods are used to only approximate the optimal value of a_k in a way that few trial steps are required to obtain a reasonable decrease in the function value.

Optimization methods are divided into different classes depending on how the search direction $\boldsymbol{p_k}$ is calculated. In this respect, BFGS is a quasi-Newton method. Newton's methods have as their starting point the second order Taylor's expansion of $f(\boldsymbol{x})$ around the current iterate $\boldsymbol{x_k}$:

$$f(\boldsymbol{x_k} + \boldsymbol{s}) \approx f(\boldsymbol{x_k}) + \boldsymbol{s}^T \boldsymbol{\nabla} f(\boldsymbol{x})|_{\boldsymbol{x_k}} + \frac{1}{2} \boldsymbol{s}^T H(\boldsymbol{x_k}) \boldsymbol{s} + \mathcal{O}\left(s^3\right) \tag{2.4}$$

where $H(\boldsymbol{x_k})$ is the Hessian matrix at point $\boldsymbol{x_k}$:

$$H(\boldsymbol{x_k}) = \left. \frac{\partial^2 f}{\partial \boldsymbol{x} \partial \boldsymbol{x}^T} \right|_{\boldsymbol{x_k}} \tag{2.5}$$

and is symmetric by Clairaut's theorem, since f is considered twice continuously differentiable. This means that the model function which is being optimized is:

$$m_k(\boldsymbol{s}) = f(\boldsymbol{x_k}) + \boldsymbol{s}^T \boldsymbol{\nabla} f(\boldsymbol{x})|_{\boldsymbol{x_k}} + \frac{1}{2}\boldsymbol{s}^T H(\boldsymbol{x_k})\boldsymbol{s} \tag{2.6}$$

which is an accurate approximation of the true f in a neighborhood close to $\boldsymbol{x_k}$.

The optimal value for $\boldsymbol{s}$ needs to be found in order to reach the local minimum around $\boldsymbol{x_k}$. For an $\boldsymbol{s} = \boldsymbol{s_k}$ to be a descent direction at $\boldsymbol{x_k}$, it has to point opposite to the direction of gradient increase:

$$\boldsymbol{s_k}^T \boldsymbol{\nabla} f(\boldsymbol{x})|_{\boldsymbol{x_k}} < 0 \tag{2.7}$$

At the local minimum $\boldsymbol{s} = \boldsymbol{s_{e,k}}$, $\nabla m_k = \boldsymbol{0}$ which leads to the optimal search direction:

$$\boldsymbol{s_{e,k}} = -H^{-1}(\boldsymbol{x_k}) \nabla f(\boldsymbol{x})|_{\boldsymbol{x_k}} \tag{2.8}$$

This is a descent direction for m_k for any $\boldsymbol{x_k} \in \mathbb{R}^{m_0}$ if Eq. (2.7) holds, and therefore if:

$$H(\boldsymbol{x}) > 0 \tag{2.9}$$

i.e. the Hessian is positive definite [NW99, 23]. Eq. (2.8) can be used directly to compute the correct search direction $\boldsymbol{p_k}$ of Eq. (2.3), with the step length $a_k = 1$.

The discussion would end here if the Hessian or its inverse were easy to compute. However, for tensor network applications computing second derivatives is expensive, and for this reason quasi-Newton methods are preferred, which instead use an approximation of the Hessian based only on gradient information. The Hessian is approximated with a symmetric, non-singular, positive definite $m_0 \times m_0$ matrix $H_k \approx H(\boldsymbol{x_k})$. It incorporates curvature information from the last two gradient computations by imposing that at $\boldsymbol{x_{k-1}} = \boldsymbol{x_k} - \boldsymbol{s_k}$, $m_k(-s_k)$ has the correct gradient [NW99, 195]:

$$\begin{aligned} \nabla m_k(\boldsymbol{s})|_{-\boldsymbol{s_k}} = \nabla f(\boldsymbol{x})|_{\boldsymbol{x_{k-1}}} \Rightarrow \nabla f(\boldsymbol{x})|_{\boldsymbol{x_k}} - H_k \boldsymbol{s_k} = \nabla f(\boldsymbol{x})|_{\boldsymbol{x_{k-1}}} \Rightarrow \\ \Rightarrow \boldsymbol{s_k} = H_k^{-1} \boldsymbol{y_k} \end{aligned} \tag{2.10}$$

where the following definitions are adopted:

$$\begin{aligned} \boldsymbol{s_k} &= \boldsymbol{x_k} - \boldsymbol{x_{k-1}}, \qquad k > 0 \\ \boldsymbol{y_k} &= \nabla f(\boldsymbol{x})|_{\boldsymbol{x_k}} - \nabla f(\boldsymbol{x})|_{\boldsymbol{x_{k-1}}}, \qquad k > 0 \end{aligned} \tag{2.11}$$

From all the matrices that satisfy condition (2.10), the BFGS algorithm chooses H_{k+1}^{-1} from H_k^{-1} as [NW99, 197]:

$$H_{k+1}^{-1} = \underset{\left\{H^{-1}:\, {}^{H^{-1}\boldsymbol{y_k} = \boldsymbol{s_k},}_{H^{-1} = \left(H^{-1}\right)^T}\right\}}{\operatorname{argmin}} \left\| H^{-1} - H_k^{-1} \right\|_{WF} \tag{2.12}$$

where WF refers to the weighted Frobenius norm:

$$\|M\|_{WF} = \left\| W^{1/2}\, M\, W^{1/2} \right\|_F \tag{2.13}$$

Other quasi-Newton algorithms arise from different choices of norm, while imposing always the condition that the inverse Hessian at the new step is as close as possible to the one at the previous step, with respect the chosen norm. To obtain an exact expression, the weight can be chosen as the average Hessian:

$$W = \int_0^1 \nabla^2 f(\boldsymbol{x})|_{\boldsymbol{x_k}+\tau \boldsymbol{s_k}} \mathrm{d}\tau \tag{2.14}$$

however other choices are acceptable as long as $Ws_k = y_k$. With this choice, the BFGS algorithm is characterized by the following inverse-Hessian update [NW99, p. 197–198]:

$$\begin{gathered} H_{k+1}^{-1} = (I - \rho_k \boldsymbol{s_k} \boldsymbol{y_k}^T) H_k^{-1} (I - \rho_k \boldsymbol{y_k} \boldsymbol{s_k}^T) + \rho_k \boldsymbol{s_k} \boldsymbol{s_k}^T \\ \text{with} \quad \rho_k = \frac{1}{\boldsymbol{y_k}^T \boldsymbol{s_k}} \end{gathered} \tag{2.15}$$

For the initial choice H_0^{-1} there is no explicit formula; it can be approximated via AD for second derivatives or finite differences, or it can also just be set to a positive definite matrix. For the benchmarks of this thesis, it was chosen identity, which proved sufficient for the problems at hand.

Under this approximation, the search direction, using Eq. (2.8) is:

$$\boldsymbol{p_k} = -H_k^{-1} \nabla f(\boldsymbol{x})|_{\boldsymbol{x_k}} \tag{2.16}$$

Now the optimal step length a_k might be different than 1, but 1 might (and usually is) a good starting point. In order to calculate the optimal step length, a line search algorithm is needed. This is the last piece of the puzzle to complete the BFGS procedure.

2.1.2 Line search: Armijo backtracking

The goal is to find a_k such that at the point $\boldsymbol{x_{k+1}}$ the function value has decreased. In addition, a_k shouldn't be chosen too small so that there is visible progress at each iteration. The algorithm that takes this condition into account is called *Armijo backtracking* [NW99, 41–42]. Backtracking starts with a trial step length $a = a_k^{(0)}$ and the *sufficient-decrease condition* (also known as *Armijo condition*) is tested:

$$f(\boldsymbol{x_k} + a\boldsymbol{p_k}) \leq f(\boldsymbol{x_k}) + c_1 a \boldsymbol{p_k}^T \nabla f(\boldsymbol{x})|_{\boldsymbol{x_k}}, \qquad 0 < c_1 < 1 \tag{2.17}$$

which states that a new accepted value of f should be below the current value by a finite amount, which depends on the step length and the gradient projection along the search direction. Typical values are $c_1 = 10^{-4}$ [NW99, 38] and $a_k^{(0)} = 1$[NW99, 42], but should be best chosen by the user for each application. If this condition doesn't hold for the initial step $a_k^{(0)}$, the trial step is updated as:

$$a_k^{(l)} \to a_k^{(l+1)} = r a_k^{(l)}, \qquad 0 < r < 1 \tag{2.18}$$

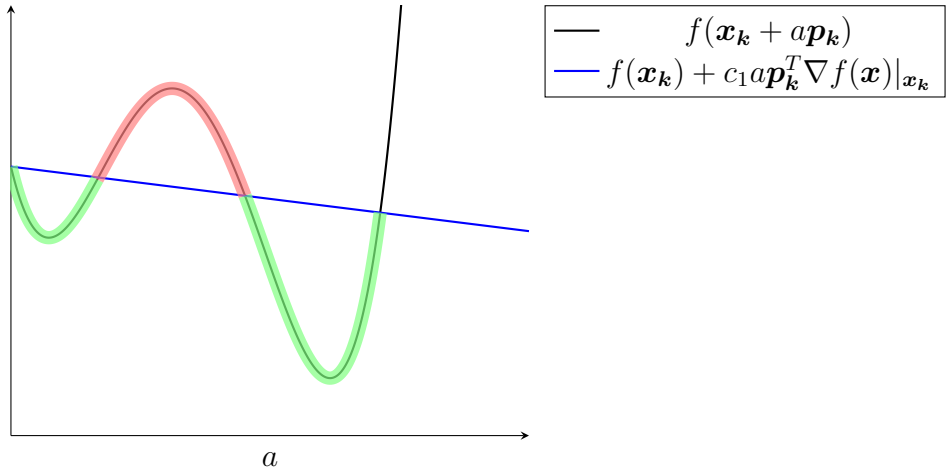


Figure 2.1: Geometric picture of sufficient decrease condition. The green area is all the points of f where the decreease is enough according to Eq. (2.17). The red are the areas where either there is an increase, or the decrease is too small.

An alternative choice is for the new trial step length to be calculated via, for example, quadratic interpolation, with the requirement that the new step is within a given region $a_k^{(l+1)} \in [a_{min}, a_{max}]$ with $0 < a_{min} < a_{max} < 1$. The minimum accepted value a_{min} shouldn't be too small so that the super-linear convergence of Newton-type methods can be exploited. With appropriate initialization $\boldsymbol{x} = \boldsymbol{x_0}$ not far from the minimum, requirement (2.17) can be enough for the optimization to progress. Nonetheless, optimization might be more efficient if an additional condition is imposed for sufficiently large curvature at the new point compared to the previous point. This condition is called second Wolfe condition [NW99, 38] and it also guarantees the positive definiteness (2.9) of the Hessian, which is not guaranteed by Eq. (2.12). However, it requires to compute the gradient at each trial point $\boldsymbol{x_k} + a_k^{(l)} \boldsymbol{p_k}$. Due to this complication, in this thesis, the second Wolfe condition was avoided, since it requires additional gradient calculations, thus slowing down the computation. Instead, as we will see in the next section, the algorithm was initialized at a point not too far from the minimum, and the Armijo backtracking was proven sufficient.

2.1.3 Limited-memory BFGS

The Limited-memory BFGS algorithm, or L-BFGS [NW99, 224], is a slight modification of the BFGS algorithm. In BFGS, H_k^{-1} as given by Eq. (2.15) depends on H_0^{-1} and $\{(\boldsymbol{s_{k-1}}, \boldsymbol{y_{k-1}}), (\boldsymbol{s_{k-2}}, \boldsymbol{y_{k-2}}), ..., (\boldsymbol{s_0}, \boldsymbol{y_0})\}$. This calculation might be computationally and memory inefficient, as in general the input space $\mathbb{R}^{m_0}$ is of large dimension and performing operations with a dense $m_0 \times m_0$ matrix can be expensive. L-BFGS was invented to make a more efficient version of the BFGS algorithm. L-BFGS has two differences with respect to BFGS:

1. At each step, the inverse Hessian is approximated by taking into account the m last iterations, where m is the *history size*, which means that H_k^{-1} is determined via recursion (2.15) only from the sets $\{(\boldsymbol{s_{k-1}}, \boldsymbol{y_{k-1}}), (\boldsymbol{s_{k-2}}, \boldsymbol{y_{k-2}}), ..., (\boldsymbol{s_{k-m}}, \boldsymbol{y_{k-m}})\}$

2. The initial guess $H_k^{-1(0)}$ of the Hessian at each iteration k can be chosen to vary. A common choice for $H_k^{-1(0)}$ is [NW99, 226]:

$$H_k^{-1(0)} = \frac{\boldsymbol{s_{k-1}^T y_{k-1}}}{\boldsymbol{y_{k-1}^T y_{k-1}}} I \tag{2.19}$$

   This choice enssures that the search direction $\boldsymbol{p_k}$ given by Eq. (2.8) is well-scaled, and the step length $a_k = 1$ is for most iterations a valid choice.

The search direction can then be computed via:

$$\begin{aligned} \boldsymbol{p_k} = &- \prod_{i=k-1}^{k-m} (I - \rho_i \boldsymbol{s_i y_i^T}) H_k^{-1(0)} \prod_{j=k-m}^{k-1} (I - \rho_j \boldsymbol{y_j s_j^T}) \nabla f(\boldsymbol{x_k}) - \\ &- \left(\sum_{i=k-m}^{k-2} \prod_{j=k-1}^{i+1} (I - \rho_j \boldsymbol{s_j y_j^T}) \rho_i \boldsymbol{s_i s_i^T} \prod_{l=i+1}^{k-1} (I - \rho_l \boldsymbol{y_l s_l^T}) + \rho_{k-1} \boldsymbol{s_{k-1} s_{k-1}^T} \right) \nabla f(\boldsymbol{x_k}) \end{aligned} \tag{2.20}$$

which comes from the recursion (2.15) by going only back to the $(k-m)$-th iteration. The algorithm that computes this matrix-vector product is called *two-loop recursion* because it first calculates the second product of the first term in one loop, and subsequently the rest via a second loop. The two-loop recursion is demonstrated in the pseudocode of Algorithm 1.

Algorithm 1: Two-loop recursion

Input: $\nabla f(\boldsymbol{x_k})$, $\{\boldsymbol{s_i}\}$, $\{\boldsymbol{y_i}\}$
Output: $\boldsymbol{p}$
$\boldsymbol{q} \leftarrow \nabla f(\boldsymbol{x_k})$;
for $i = k-1, k-2, ..., k-m$ **do**
  $a_i \leftarrow \rho_i \boldsymbol{s_i^T q}$;
  $\boldsymbol{q} \leftarrow \boldsymbol{q} - a_i \boldsymbol{y_i}$;
end
$\boldsymbol{r} \leftarrow H_k^{-1(0)} \boldsymbol{q}$, where $H_k^{-1(0)}$ is calculated via Eq. (2.19);
for $i = k-m, k-m+1, ..., k-1$ **do**
  $b \leftarrow \rho_i \boldsymbol{y_i^T r}$;
  $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{s_i}(a_i - b)$;
end
$\boldsymbol{p} \leftarrow -\boldsymbol{r}$;

When $m \times m_0 \sim m_0^2$, L-BFGS becomes inefficient compared to plain BFGS in terms of computation and storage requirements, since in the two-loop recursion many operations

would be repeated for each iteration, and m vectors of length m_0 would have to be stored in memory. For the problems that we consider however, $m_0 >> 1$ and $m \sim 10$, therefore L-BFGS is the most suitable choice. In Algorithm 2 the summary of the whole L-BFGS procedure described in this subsection is presented in the form of a pseudocode.

Algorithm 2: L-BFGS

Input: $\boldsymbol{x_0} \in \mathbb{R}^{m_0}$, $m \in \mathbb{N}_{>0}$, a_0, $c_1 \in (0,1)$, ConvergenceCriteria
Output: $\boldsymbol{x_e}$ minimizer of $f(\boldsymbol{x})$
$H_0^{-1} = I_n$; `/* Initial inverse Hessian set, e.g., to identity */`
$\boldsymbol{p_0} \leftarrow -H_0^{-1}\nabla f(\boldsymbol{x_0})$;
$k = 0$;
while *True* **do**
  $a_{tmp} \leftarrow a_0$;
  $\boldsymbol{x_{tmp}} \leftarrow \boldsymbol{x_k} + a_{tmp}\boldsymbol{p_k}$;
  while $f(\boldsymbol{x_{tmp}}) > f(\boldsymbol{x_k}) + c_1 a_{tmp}\boldsymbol{p_k}^T \nabla f(\boldsymbol{x})|_{\boldsymbol{x_k}}$ **do**
    $a_{tmp} \leftarrow a_{tmp}/2$; `/* Alternative choices indicated in the text */`
    $\boldsymbol{x_{tmp}} \leftarrow \boldsymbol{x_k} + a_{tmp}\boldsymbol{p_k}$;
  end
  $\boldsymbol{x_k} \leftarrow \boldsymbol{x_{tmp}}$;
  $\boldsymbol{s_k} \leftarrow a_{tmp}\boldsymbol{p_k}$ and save;
  $\boldsymbol{y_k} \leftarrow \nabla f(\boldsymbol{x_k}) - \nabla f(\boldsymbol{x_{k-1}})$ and save;
  $k \leftarrow k + 1$;
  if $k > m$ **then**
    discard $\boldsymbol{s_{k-m-1}}$, $\boldsymbol{y_{k-m-1}}$;
  end
  if *ConvergenceCriteria* **then**
    break;
  end
  Update $\boldsymbol{p_k}$ with Algorithm 1;
end
$\boldsymbol{x_e} \leftarrow \boldsymbol{x_k}$

2.1.4 Generalization to functions of complex variables

For tensor network applications, it is important to have a generalization of the optimization procedure to complex variables. The problem we consider is having a function $f : \mathbb{C}^{m_0} \to \mathbb{R}$ and searching the solution to:

$$\min_{\boldsymbol{z} \in \mathbb{C}^{m_0}} f(\boldsymbol{z}) \tag{2.21}$$

In Sec. 1.1 we saw that, through automatic differentiation, the Wirtinger derivatives are computed (up to a factor of two), which are related to each other through

$$\frac{\partial f_k}{\partial \boldsymbol{z}^*} = \left(\frac{\partial f_k}{\partial \boldsymbol{z}}\right)^* \tag{2.22}$$

because $f(\boldsymbol{z}) \in \mathbb{R}$. Therefore, it is desirable to adapt the L-BFGS algorithm so that it uses these complex derivatives. The difficulty lies in the fact that a non-constant function f with range in the real numbers is non-analytic in $\mathbb{C}^{m_0}$, as can be proved using the Cauchy-Riemann equations. However, a generalization of the Taylor series does exist, and is used in [SvBdL12] in the context of optimization of real functions in the complex domain.

Suppose that f is twice continuously differentiable in the space spanned by $\tilde{\boldsymbol{x}} = (\boldsymbol{x}, \boldsymbol{y}) = (\operatorname{Re} \boldsymbol{z}, \operatorname{Im} \boldsymbol{z}) \in \mathbb{R}^{2m_0}$. Then, Eq. (2.4) holds for $f(\tilde{\boldsymbol{x}})$. The variables z and z^* generally span different spaces. Gradient-based optimization should function in a way that has the whole derivative information, and for this, in general, the derivatives with respect to both variables are needed. Because the space spanned by $\tilde{\boldsymbol{z}} = (\boldsymbol{z}, \boldsymbol{z}^*)$ is isomorphic to the space spanned by $\tilde{\boldsymbol{x}}$, f inherits the same analyticity properties in the space spanned by $\tilde{\boldsymbol{z}}$. Then, the only part of the previous section that needs reformulation is the Taylor expansion (2.4), which needs to be written with respect to $\tilde{\boldsymbol{z}}$. The model function that has to be optimized by Newton's methods is:

$$m_k(\tilde{\boldsymbol{s}}) = f(\tilde{\boldsymbol{z}}_{\boldsymbol{k}}) + \tilde{\boldsymbol{s}}^\dagger \left.\frac{\partial f}{\partial \tilde{\boldsymbol{z}}^*}\right|_{\tilde{\boldsymbol{z}}_{\boldsymbol{k}}} + \frac{1}{2}\tilde{\boldsymbol{s}}^\dagger H(\tilde{\boldsymbol{z}}_{\boldsymbol{k}})\tilde{\boldsymbol{s}} \tag{2.23}$$

where now the second order complex Taylor series was used and the complex Hessian is [vdB94]:

$$H(\tilde{\boldsymbol{z}}_{\boldsymbol{k}}) = \left.\frac{\partial^2 f}{\partial \tilde{\boldsymbol{z}}^* \partial \tilde{\boldsymbol{z}}^T}\right|_{\tilde{\boldsymbol{z}}_{\boldsymbol{k}}} \tag{2.24}$$

Defined in this way, the Hessian is not symmetric but a Hermitian matrix, due to Eq. (2.22). The search direction is then computed by setting $\nabla m_k(\tilde{\boldsymbol{s}}_{\boldsymbol{e},\boldsymbol{k}}) = \boldsymbol{0}$, where the complex gradient is defined in Eq. (1.12). In this way, the expression is completely analogous to the real case:

$$\tilde{\boldsymbol{s}}_{\boldsymbol{e},\boldsymbol{k}} = -H^{-1}(\tilde{\boldsymbol{z}}_{\boldsymbol{k}}) \nabla f(\tilde{\boldsymbol{z}})|_{\tilde{\boldsymbol{z}}_{\boldsymbol{k}}} \tag{2.25}$$

and it has to point opposite to $\nabla f(\tilde{\boldsymbol{z}})|_{\tilde{\boldsymbol{z}}_k}$, analogously to Eq. (2.7). The rest of the results are completely analogous to the real case under this framework [SvBdL12][1]; for example, in the context of BFGS, the approximation H_k^{-1} should now be a Hermitian (positive definite) matrix, and the resulting expression is also analogous to the real case:

$$\begin{gathered} H_{k+1}^{-1} = (I - \rho_k^{(c)} \tilde{\boldsymbol{s}}_{\boldsymbol{k}} \tilde{\boldsymbol{y}}_{\boldsymbol{k}}^\dagger) H_k^{-1} (I - \rho_k^{(c)} \tilde{\boldsymbol{y}}_{\boldsymbol{k}} \tilde{\boldsymbol{s}}_{\boldsymbol{k}}^\dagger) + \rho_k^{(c)} \tilde{\boldsymbol{s}}_{\boldsymbol{k}} \tilde{\boldsymbol{s}}_{\boldsymbol{k}}^\dagger \\ \text{with} \quad \rho_k^{(c)} = \frac{1}{\tilde{\boldsymbol{y}}_{\boldsymbol{k}}^\dagger \tilde{\boldsymbol{s}}_{\boldsymbol{k}}} \end{gathered} \tag{2.26}$$

[1] Some results differ by a factor of 2 with respect to results of [SvBdL12] due to our use of Eq. (1.12) for the gradient, which is consistent with the AD calculation. This allows for the use of the standard initial trial step length $a_k^{(0)} = 1$ also in this case.

The search direction $\boldsymbol{p_{k+1}} = -H_{k+1}^{-1}\nabla f(\tilde{\boldsymbol{z}})|_{\tilde{\boldsymbol{z}}_{\boldsymbol{k+1}}}$ can be written in a simpler way by using that:

$$\rho_k^{(c)}\tilde{\boldsymbol{a}}\tilde{\boldsymbol{b}}^\dagger\tilde{\boldsymbol{c}} = \rho_k \,\mathrm{Re}(\boldsymbol{b}^\dagger\boldsymbol{c})\boldsymbol{a}, \qquad \boldsymbol{a},\boldsymbol{b},\boldsymbol{c}\in\mathbb{C}^n \tag{2.27}$$

with

$$\rho_k = \frac{1}{\mathrm{Re}(\boldsymbol{y}_{\boldsymbol{k}}^\dagger\boldsymbol{s}_{\boldsymbol{k}})} \tag{2.28}$$

Algorithm 3 gives the two-loop recursion that computes $\boldsymbol{p_k}$.

Algorithm 3: Two-loop recursion (complex variables)

Input: $\nabla f(\boldsymbol{z_k})$, $\{\boldsymbol{s_i}\}$, $\{\boldsymbol{y_i}\}$
Output: $\boldsymbol{p}$
$\boldsymbol{q} \leftarrow \nabla f(\boldsymbol{z_k})$;
for $i = k-1, k-2, ..., k-m$ **do**
  $a_i \leftarrow \rho_i \,\mathrm{Re}(\boldsymbol{s_i^\dagger q})$, ρ_i from Eq. (2.28);
  $\boldsymbol{q} \leftarrow \boldsymbol{q} - a_i\boldsymbol{y_i}$;
end
$\boldsymbol{r} \leftarrow H_k^{-1(0)}\boldsymbol{q}$ where $H_k^{-1(0)}$ is calculated via Eq. (2.30);
for $i = k-m, k-m+1, ..., k-1$ **do**
  $b \leftarrow \rho_i \,\mathrm{Re}(\boldsymbol{y_i^\dagger r})$;
  $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{s_i}(a_i - b)$;
end
$\boldsymbol{p} \leftarrow -\boldsymbol{r}$;

The line search is done with updates of the form:

$$\boldsymbol{z}_k \rightarrow \boldsymbol{z_{k+1}} = \boldsymbol{z_k} + a_k\boldsymbol{p_k^*}, \qquad a_k \in \mathbb{R} \tag{2.29}$$

Finally, regarding L-BFGS, the generalization of Eq. (2.19) is:

$$H_k^{-1(0)} = \frac{\mathrm{Re}(\boldsymbol{s_{k-1}^\dagger y_{k-1}})}{\boldsymbol{y_{k-1}^\dagger y_{k-1}}} I \tag{2.30}$$

In this way, the search for a minimum of $f(\boldsymbol{z})$ can be done in a compact way for real and complex variables, by use of the Wirtinger derivatives. Algorithm 4 gives the complete L-BFGS procedure.

2.2 Initialization of tensor network

Algorithms like the BFGS converge locally rather than globally, which means that the initialization is an important factor which might determine the solution. For this reason, it is important to be equipped with an initial state adequately close to the minimum, and possibly actively prevent the algorithm from getting stack at local minima. In this section, these issues are addressed for the case of tensor networks.

Algorithm 4: L-BFGS (complex variables)

Input: $\boldsymbol{z_0} \in \mathbb{C}^{m_0}$, $m \in \mathbb{N}_{>0}$, a_0, $c_1 \in (0,1)$, ConvergenceCriteria
Output: $\boldsymbol{z_e}$ minimizer of $f(\boldsymbol{z})$
$H_0^{-1} = I_n$; `/* Initial inverse Hessian set, e.g., to identity */`
$\boldsymbol{p_0} \leftarrow -H_0^{-1} \nabla f(\boldsymbol{z_0})$;
$k = 0$;
while *True* **do**
  $a_{tmp} \leftarrow a_0$;
  $\boldsymbol{z_{tmp}} \leftarrow \boldsymbol{z_k} + a_{tmp} \boldsymbol{p_k^*}$;
  while $f(\boldsymbol{z_{tmp}}) > f(\boldsymbol{z_k}) + c_1 a_{tmp} \mathrm{Re}(\boldsymbol{p_k^\dagger} \nabla f(\boldsymbol{z})|_{\boldsymbol{z_k}})$ **do**
    $a_{tmp} \leftarrow a_{tmp}/2$; `/* Alternative choices indicated in the text */`
    $\boldsymbol{z_{tmp}} \leftarrow \boldsymbol{z_k} + a_{tmp} \boldsymbol{p_k^*}$;
  end
  $k \leftarrow k + 1$;
  $\boldsymbol{z_k} \leftarrow \boldsymbol{z_{tmp}}$;
  $\boldsymbol{s_k} \leftarrow a_{tmp} \boldsymbol{p_k^*}$ and save;
  $\boldsymbol{y_k} \leftarrow \nabla f(\boldsymbol{z_k}) - \nabla f(\boldsymbol{z_{k-1}})$ and save;
  if $k > m$ **then**
    discard $\boldsymbol{s_{k-m-1}}$, $\boldsymbol{y_{k-m-1}}$;
  end
  if *ConvergenceCriteria* **then**
    break;
  end
  Update $\boldsymbol{p}$ with Algorithm 3;
end
$\boldsymbol{z_e} \leftarrow \boldsymbol{z_k}$

Simple update

For tensor networks a relatively good guess of the ground state can be approximated via algorithms based on imaginary time evolution [Vid07]. The computationally cheapest solution is to use a simple update algorithm [JWX08] which is used to initialize the benchmarks of this thesis. A summary of the simple update algorithm is presented below.

As discussed in Sec. 1.2.1 we are interested in local Hamiltonians of the form:

$$\mathcal{H} = \sum_{i,j \in n_i} h_{i,j} \tag{2.31}$$

Imaginary time evolution is used to approximate ground states for such Hamiltonians using an imaginary time unit $\beta = it$. The decomposition of the time-evolution operator in its eigenbasis:

$$e^{-\beta \mathcal{H}} = \sum_i e^{-\beta E_i} |i\rangle \langle i| \tag{2.32}$$

at the limit $\beta \to \infty$ tends to maintain only contributions from the ground state. Updating the whole supercell with this gate is practically impossible. For this reason it is common to apply time evolution gates for small time intervals $\delta\tau = \beta/N$:

$$e^{-\beta \mathcal{H}} = \left(e^{-\delta\tau \mathcal{H}}\right)^N \tag{2.33}$$

for $\delta\tau \to 0$ and $N \to \infty$, and to use the first order Trotter decomposition:

$$e^{-\delta\tau \mathcal{H}} \approx \prod_{i,j \in n_i} e^{-\delta\tau h_{i,j}} + \mathcal{O}\left(\delta\tau^2\right) \tag{2.34}$$

In this way, only two-site gates $e^{-\delta\tau h_{i,j}}$ are applied. This decomposition ignores the possible non-commutativity of different terms $h_{i,j}$, thus introducing an error $\mathcal{O}(\delta\tau^2)$ known as Trotter error.

For imaginary time evolution, usually the $\Gamma - \Lambda$ gauge is used. This gauge is introduced in App. A for a MPS starting from a canonical form, and we see there that the right and left bond tensors Λ contain all the information regarding the right and left environment of that tensor. The fact that the bond tensors are used to simulate the environment is accurate only for tree-like networks because cutting a bond of the network divides it into two parts and pure-state bipartite entanglement characterizes the correlations between the left and right part. In the case of square PEPS such a gauge can also be employed as in Fig. 2.2, but only information from a small region around the tensors is contained in the bond tensors. In Fig. 2.2 the relation of the $\Gamma - \Lambda$ gauge to the gauge used in CTMRG is depicted.

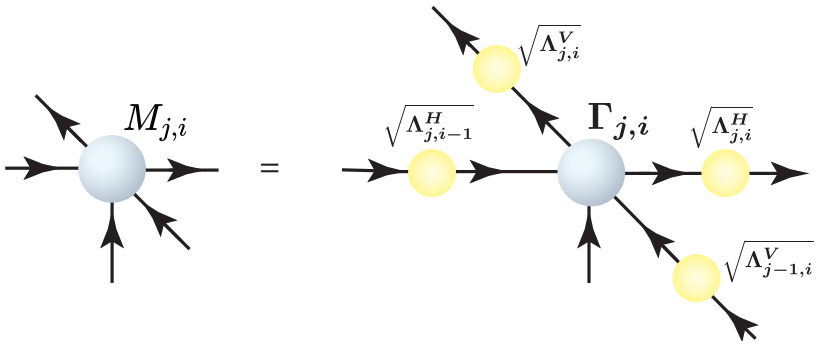


Figure 2.2: Relation of the gauge used for the CTMRG with respect to the gauge used for the simple update.

The steps of one simple update step are shown diagrammatically in Fig. 2.3 for nearest-neighbor interactions. We start by applying a Trotter gate $e^{-\delta\tau h_{(j,i),(j,i+1)}}$ to sites $(j,i)-(j,i+1)$, thus increasing the dimension of the intermediate bond between the two sites to D^3d. Then, through an SVD and truncation the bond dimension can be decreased back to D. A further rescaling is needed to bring the tensor network back to its $\Gamma-\Lambda$ form. This must be repeated for each pair of interacting sites in the supercell. The starting Γ and Λ tensors are chosen randomly, and the time-evolution gates are applied, until convergence of the local energy expectation values, such as the one shown in Fig. 2.4. Next-nearest neighbor terms between (anti-)diagonally located sites can be updated in a similar fashion (see, for example, the implementation in [Zha21] for further details).

Finally, it is important to note that the optimization procedure described in Sec. 2.1 conserves the structure of the tensors of the tensor network. This means that the block structure of the tensors is determined by the structure of the converged tensors from the simple update algorithm, and by the symmetry which we enforce initially on the iPEPS. The optimization is restricted to this structure and can only improve on the numerical values of the tensor elements.

Gauging with Belief Propagation

For an MPS, starting from the canonical form we can consistently build the $\Gamma-\Lambda$ gauge so that the identity A.6 holds, or at least is approximated with controllable accuracy (see App. A). The equivalent of the isometric condition A.6 for a square lattice is shown in Fig. 2.5. In the following discussion, the term *Vidal gauge* will be used to describe a tensor network in the $\Gamma-\Lambda$ form that also satisfies this condition, out of Guifré Vidal who originally used it in the context of imaginary time evolution for an MPS. Identity (2.5) is nowhere explicitly imposed during the simple update steps of the square iPEPS, but we see from the example of the MPS that it is important in order to contract the tensor network and calculate expectation values of local observables such as (2.4). For general tensor network states, such as the PEPS, there is no canonical form from which to obtain the Vidal gauge, and we have to invent other ways to obtain it so that it respects the isometric condition. The simple update algorithm presented previously eventually

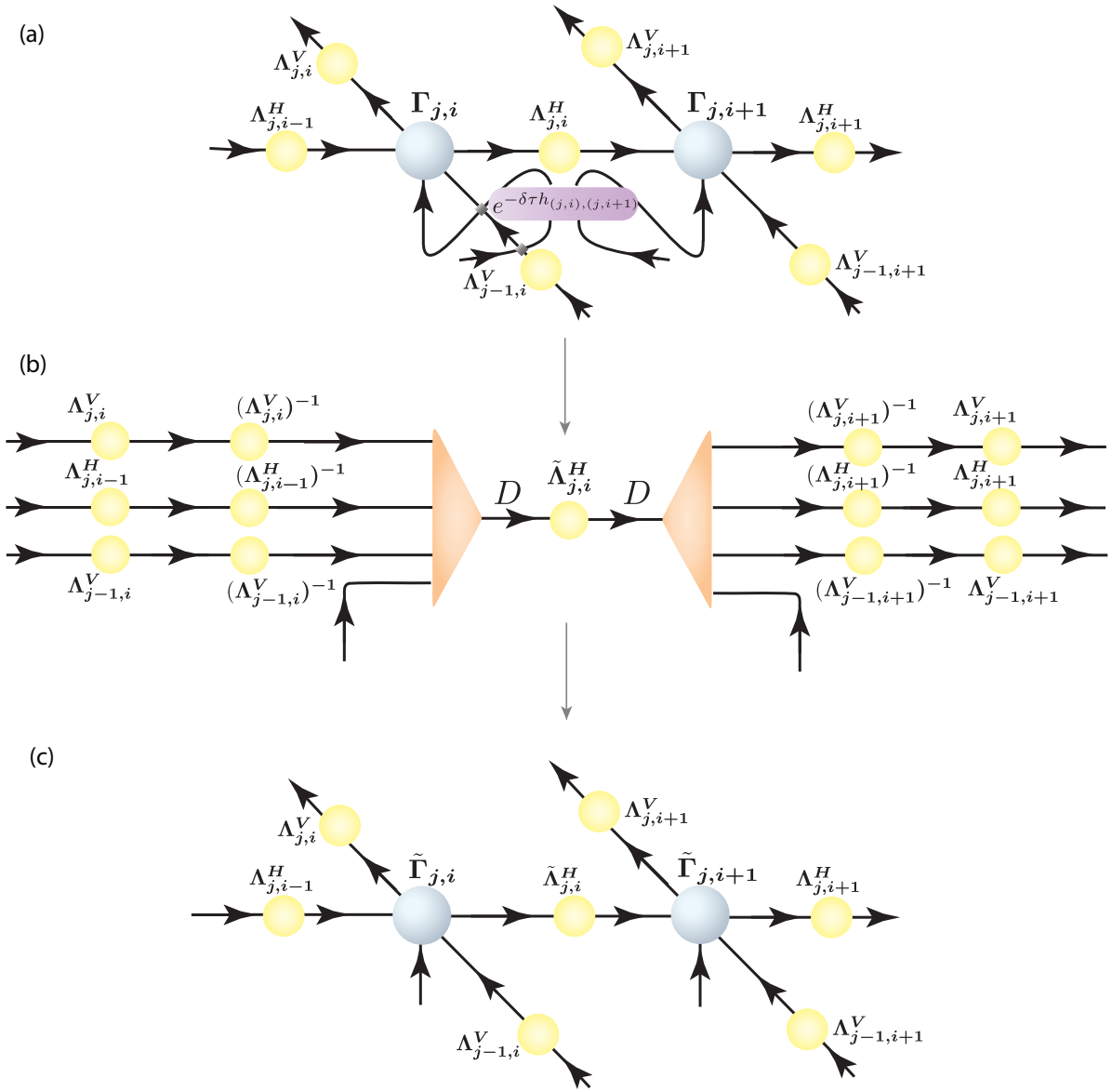


Figure 2.3: Simple update of horizontal bond $(i,j)-(i,j+1)$. (a) Update with a Trotter gate. (b) SVD and rescale the isometries with the inverses of the bond tensors. The singular value matrix defines the updated Λ tensor. (c) Define the updated Γ tensor.

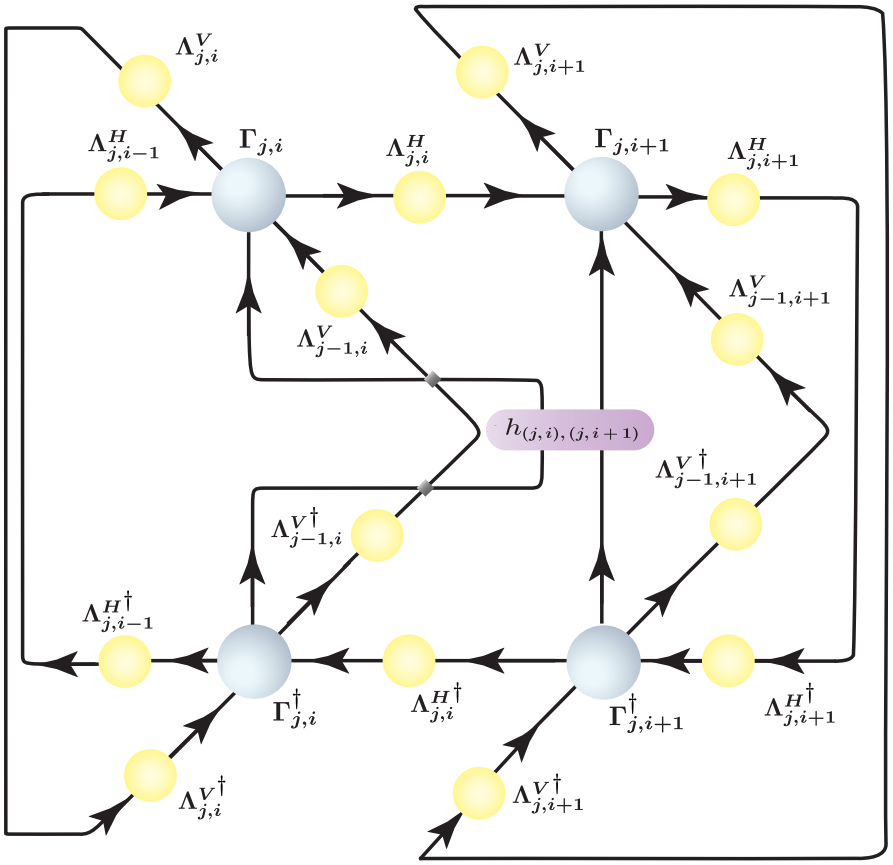


Figure 2.4: Expectation value of $h_{(j,i),(j,i+1)}$ at the $\Gamma - \Lambda$ gauge. This expectation value is not the true expectation of $h_{(j,i),(j,i+1)}$, since only sites i, j are considered.

converges to the Vidal gauge, but there is no guarantee that it is in the Vidal gauge at every step, and truncations during SVDs or due to insufficiently small time steps $\delta\tau$ prevent it from reaching exactly this gauge. A method to remedy this was recently proposed [TF23], and maintains the tensor network close to the Vidal gauge at every step of the simple update. Among other advantages, this method acts as an improvement on simple update, being characterized by faster convergence to the Vidal gauge and lower variational energies.

This new gauging technique was examined within the scope of this thesis as a possible way of pre-training the tensor network before starting the optimization based on CTMRG. Eventually, it did not prove to be a better candidate for initializing gradient-based optimization than the usual simple update, but it is presented below so that the corresponding results can be discussed in the next chapter, and because it might be useful in other contexts.

The general strategy is based on Belief Propagation (BP), a method originally used in the context of statistical inference, but which also found application in tensor networks.

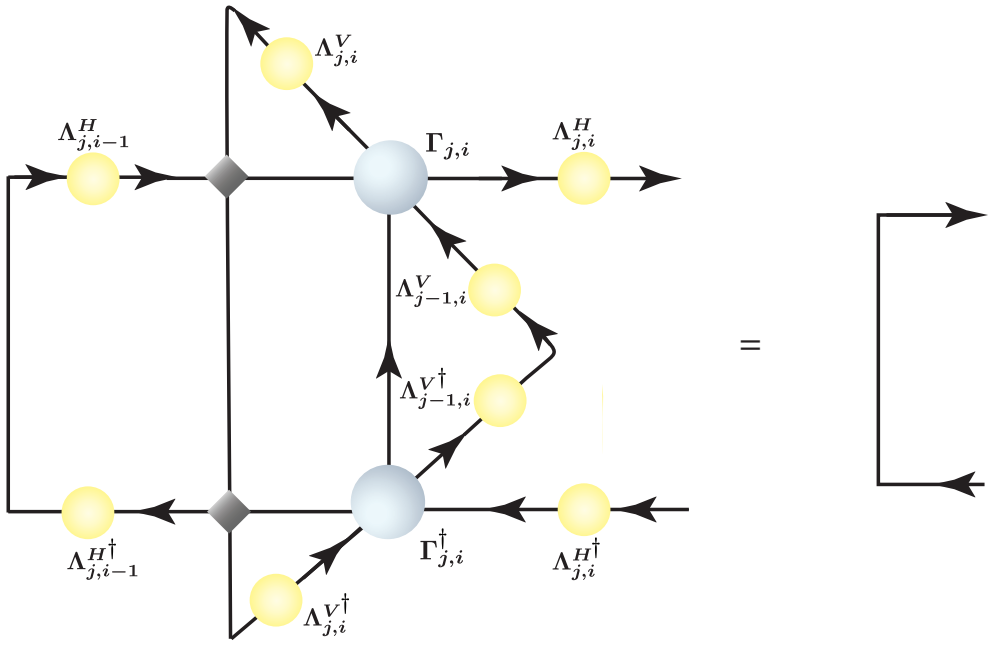


Figure 2.5: Isometric condition for a square PEPS tensor network in the Vidal gauge. Equivalent equations it holds for the other bonds.

Belief propagation is used to form a new gauge, called the BP gauge. One starts from a double-layer supercell made up of tensors $\mathcal{T}_{i,j}$ and simulates the periodicity with extra bonds, as, e.g. for a 2×2 supercell:

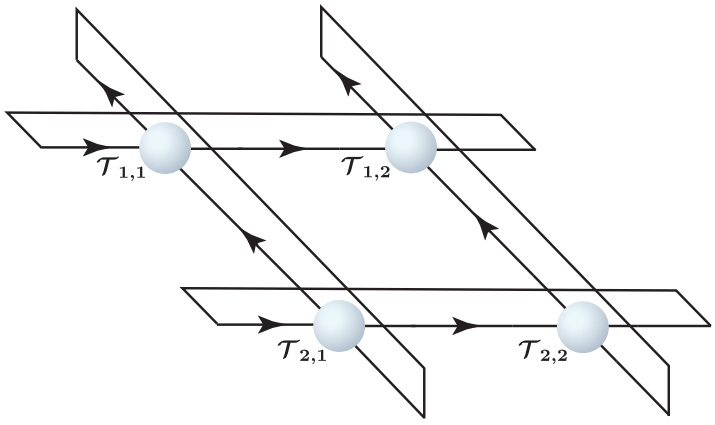


Figure 2.6: Periodic PEPS made up of the double-layer tensors of the supercell.

The environment for each tensor $\mathcal{T}_{i,j}$ is simulated via four tensors for each site called *message tensors*, $Msg^U_{j,i}$, $Msg^D_{j,i}$, $Msg^R_{j,i}$ and $Msg^L_{j,i}$, as depicted in Fig. 2.7. The labels U, D, R, L show the position of the $(j,i)-$th message tensor with respect to site (j,i). The message tensors $Msg^R_{j,i}$ are obtained via the self-consistent conditions of Fig. 2.8,

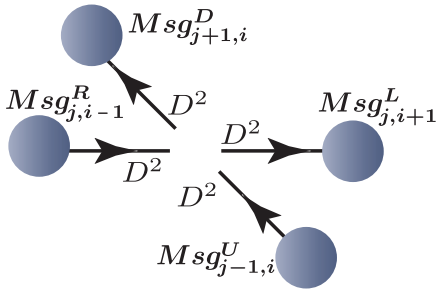


Figure 2.7: Environment of site (j, i) as given by the message tensors.

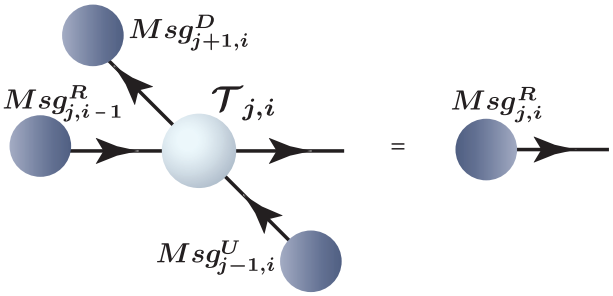


Figure 2.8: Fixed-point condition for message tensors $Msg^R_{j,i}$. The depicted tensor acts as the left environment for site $(j, i+1)$.

and equivalently for the other bonds. By iterating this self-consistent equation over the supercell tensors, one can obtain fixed-point message tensors. At the fixed point, the identity of Fig. 2.9 holds, which can be derived directly from the fixed-point conditions 2.8. These fixed-point message tensors provide a representation of part of the environment of tensor $\mathcal{M}_{i,j}$ in a similar way as the Λ tensors do in the Vidal gauge. Simple update can be implemented directly using this gauge. After each simple update step, or after a few steps, condition 2.8 needs to be iterated again until convergence.

Once the simple update has converged, the Vidal gauge can be obtained through an SVD on each of the bonds, as shown in Fig. (2.10). In this figure, the definition of the square of a matrix is defined as: $M = U^\dagger DU$, $M^{1/2} = D^{1/2}U$ and $M^{-1/2} = U^{1/2}D^{-1/2}$, or the corresponding pseudoinverse. These relations, combined with identity 2.9 guarantee the isometric condition 2.5 for the Vidal gauge, up to a small error due to truncation during the SVD and inversion of matrices.

In this way, the isometric property is restored. This results in an improved approximation of local observables such as the local energy terms of Fig. 2.4. Nevertheless, it still is a very crude approximation of the environment, and eventually it might not converge too far from the fixed point of regular simple update.

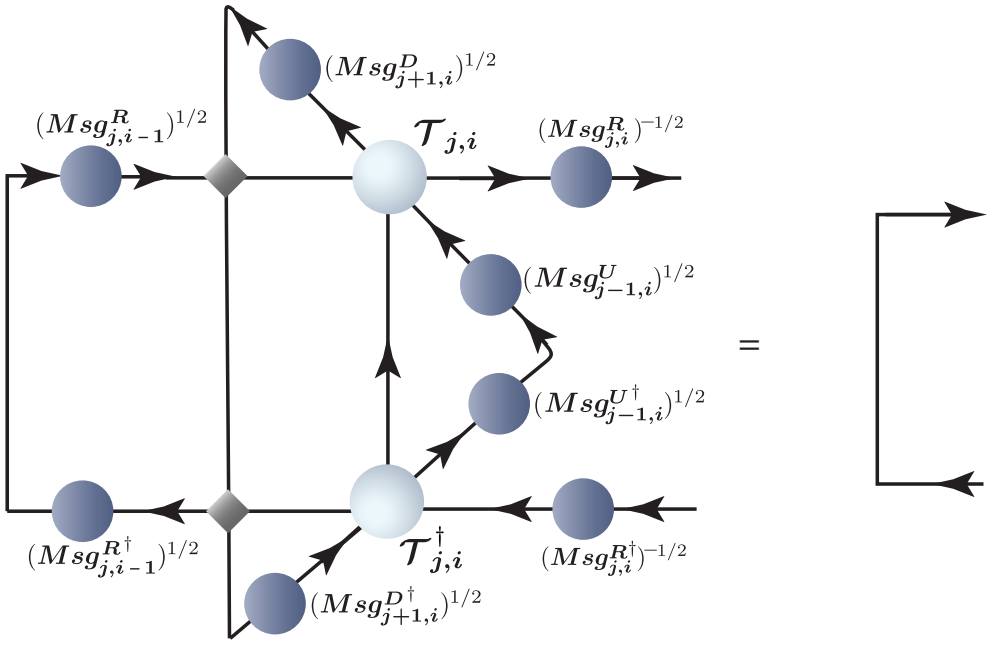


Figure 2.9: Isometric property of the message tensors at the fixed point.

Escaping local minima

As mentioned in Sec. 2.1, optimization methods might converge to local minima instead of global ones, depending on the initialization of the algorithm. Simple update is a convenient way to initialize the tensor network to start the optimization. However, it is prone to get stuck at local minima, which might prevent the optimization from reaching the global solution. One way to tackle this [NWR+24] is to perturb with small random noise ($\sim 10^{-2}$) the tensors where simple update has converged, and after multiple trials, choose the best solution.

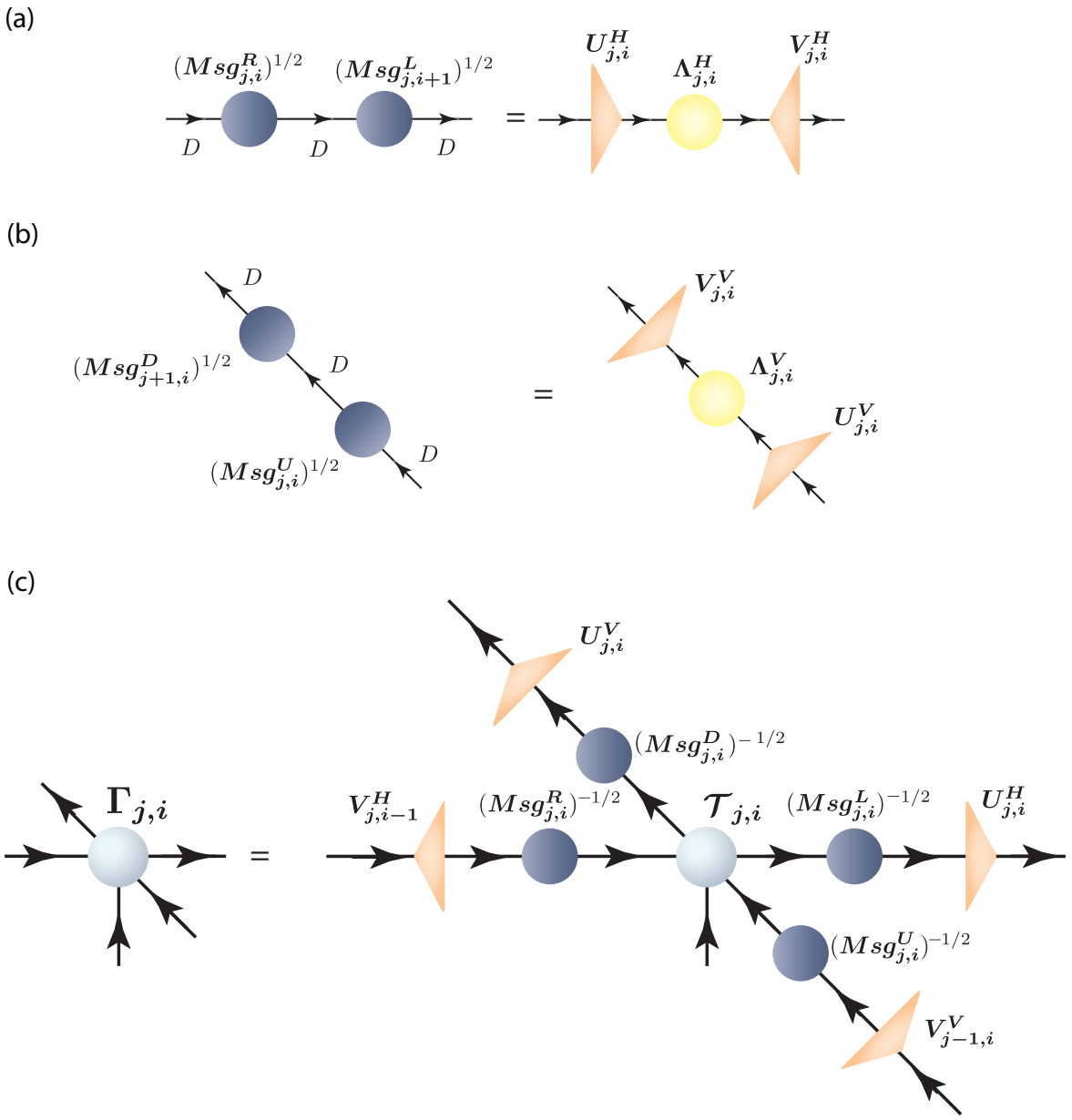


Figure 2.10: Relation of the BP gauge to the Vidal gauge. (a) A SVD to the horizontal square-root message tensors gives the horizontal bond tensors. (b) A SVD to the vertical square-root message tensors gives the vertical bond tensors. (c) Rescale $\mathcal{T}$ tensors with square-root message tensors and isometries to obtain the Γ tensors.

Chapter 3

Application to ground-state search of quantum lattice models

Models that effectively represent quantum Hamiltonians on a lattice serve as the preferred tool for examining quantum systems with strong interactions. This chapter is devoted to the simulation results of ground-state search for several quantum lattice models, using AD-based optimization of iPEPS as presented in Ch. 1 and Ch. 2. Sec. 3.1 concerns the validation of the performance of the AD/L-BFGS implementation for variational optimization of iPEPS. I present benchmark results for spin systems using the nearest-neighbor (NN) Heisenberg model, whose ground-state energy is known to high accuracy. I additionally provide results for a model of non-interacting spinful fermions featuring an exact solution for the ground-state energy. This allows for a highly accurate comparison of the performance of AD applied to fermionic iPEPS. In Sec. 3.2 the Hubbard model is addressed, starting from the Hamiltonian at half-filling and NN interaction. The performance of AD is compared with results based on the simple update algorithm. Next, the belief propagation technique described in Sec. 2.2 is employed for this case, to explore potential benefits in the search for the ground state. Finally, the same method is applied for finite doping and next-nearest neighbor (NNN) interaction for a U(1) iPEPS with period eight, and a SU(2), uniform iPEPS.

3.1 Benchmark results

The protocol followed for the AD benchmarks is as follows:

1. The iPEPS is initialized to random $\Gamma - \Lambda$ tensors with $D = 2$ and a value is defined for the environment bond dimension χ. Typically, for iPEPS with U(1) symmetry, $\chi = 10D$ is chosen in practice, and $\chi = 5D^*$ for SU(2) symmetry, where D^* is the number of multiplets kept.

2. Simple update is run, starting from time step $\delta\tau = 0.1$ and ending at $\delta\tau = 10^{-5}$ until convergence.

3. Starting from the last state of step 2, CTMRG is run to obtain the converged environmental tensors, and the expectation value of the energy is calculated. The computational graph is saved in the process. Checkpoints are used to reduce memory costs.

4. L-BFGS optimization is performed using AD, until satisfactory convergence of the variational energy. The history size is set to $m = 20$, as this was found to be adequate; increasing it did not show additional advantages. For the line search, the initial trial step length for each iteration k is chosen $a_0^{(k)} = 1$, unless otherwise stated. It is updated at every step via quadratic interpolation, considering the energy and gradient from the prior step and the energy at the previous trial point, until either the condition for sufficient decrease is fulfilled or convergence is achieved (see Sec. 2.1). An alternative solution is to decrease the step by a factor of two, which does not change the result but might find a solution more slowly.

5. The bond dimension is increased by 1, the new environmental bond dimension is defined, and we go back to step 2.

As a side-comment, the converged tensors of step 2 are in some cases perturbed with small random noise (see Sec. 2.2), and the above process is also run for multiple such perturbed states so that the optimal result can be selected. However, this step did not prove to be crucial in the context of this specific implementation, in the sense that the final optimal solution resulting from perturbing the initial tensors was located very closely to the original one.

3.1.1 The Heisenberg model

First we start with the nearest-neighbor anti-ferromagnetic Heisenberg model:

$$H = \sum_{<i,j>} \boldsymbol{S_i} \cdot \boldsymbol{S_j} \tag{3.1}$$

where $\boldsymbol{S_i}$ is the spin-1/2 operator of site i. This model captures the magnetic properties of materials by simulating the interaction of spins on a lattice. Each spin occupies one lattice site and only the nearest-neighbor interactions are non-zero.

Fig. 3.1 shows the benchmark results for the Heisenberg model, considering a U(1)-invariant iPEPS with a 2×2 supercell, and following the aforementioned protocol for bond dimensions $D = 2$ up to $D = 7$. The relative error for the variational energy is plotted, obtained using as reference the more accurate extrapolated result with the quantum Monte Carlo method [San97], $E_g = -0.669437$.[1] The results are also compared with the equivalent results based on a plane simple update. The starting state ($D = 2$) for the simple update and for the variational optimization was the same, and the results are therefore comparable.

[1]The energies here and throughout the text are dimensionless, as they are considered relative to the coupling constant.

This figure is a proof of principle for the variational optimization with AD, as it shows that the results are up to an order of magnitude improved compared to the simple update results.

For each bond dimension, the L-BFGS algorithm typically required several tens of iterations to reach convergence. Larger dimensions demanded more iterations, but generally no more than 60 were necessary. For the CTMRG to converge well, about 15 iterations were required. The computational cost of AD, measured as the ratio of the time T_f for a forward sweep over the time T_r for a reverse sweep, was in a range of approximately $\frac{T_f}{T_r} \sim 2-3$ without the use of checkpointing, and increased to $\frac{T_f}{T_r} \sim 5-7$ with checkpointing applied as described in Sec. 1.3.2.

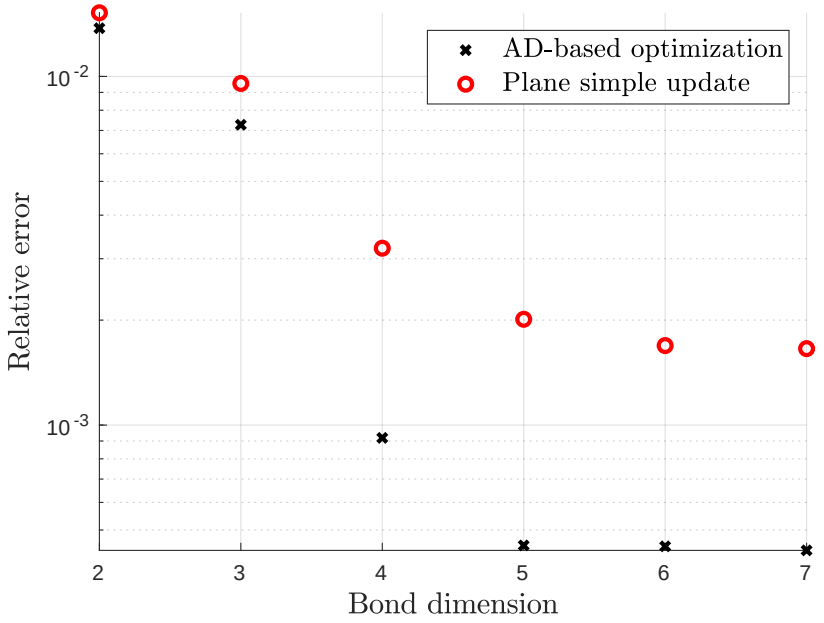


Figure 3.1: Relative error of variational energy for 2D Heisenberg anti-ferromagnet on a square lattice, with reference a quantum Monte Carlo calculation [San97], as a function of D. The data provided show the simple update results (red) and the improved results obtained via L-BFGS optimization (black). The y-axis is in logarithmic scale.

3.1.2 Free fermion model

Tensor networks provide particularly competitive algorithmic architectures for the simulation of fermionic systems, as the main competitors, quantum Monte Carlo methods, are cursed with the negative sign problem [TW05] which results in exponential scaling in computing times. For this reason, computational quantum many-body physics can profit significantly from improved tensor network calculations for strongly correlated fermions.

Before going to interacting systems, in order to test the AD implementation for fermionic iPEPS the free-fermion Hamiltonian is benchmarked:

$$H = -\sum_{<i,j>,\sigma} (c^\dagger_{i\sigma} c_{j\sigma} + c_{i\uparrow} c_{j\downarrow} - c_{i\downarrow} c_{j\uparrow} + h.c.) + \mu \sum_{i,\sigma} n_{i\sigma} \tag{3.2}$$

In the first sum, the first term and its conjugate are the nearest-neighbor hopping terms, with $c^\dagger_{i\sigma}$ being the creation operator of a fermion at site i with spin $\sigma \in \{\frac{1}{2}, -\frac{1}{2}\}$. The term $c_{j\downarrow} - c_{i\downarrow} c_{j\uparrow} + h.c.$ indicates the singlet pairing, μ is the chemical potential, and $n_{i,\sigma} = c^\dagger_{i\sigma} c_{i,\sigma}$ is the particle number operator. The ground state energy of this Hamiltonian can be computed analytically [ZLvD23], and for $0 < \mu \leq 8$ is characterized by an energy gap:

$$\Delta = \frac{\mu}{\sqrt{2}} \tag{3.3}$$

Because of this property, the larger the chemical potential the more reliable the iPEPS representation is (see Sec. 1.2.1). Therefore, to benchmark the AD for fermionic iPEPS, $\mu = 8$ is chosen, for which the analytical solution, calculated using a 100×100 finite-size lattice, gives $E_g = -0.27318687$. The supercell is again chosen 2×2 and both U(1) and SU(2) iPEPS are studied (with also $\mathbb{Z}_2$ symmetry to ensure the fermionic parity conservation explained in Sec. 1.2.4). In Fig. 3.2 the variational energy is given for iPEPS bond dimensions $D = 3, 4, 5, 6, 7$ for the case of L-BFGS optimization, as well as for the the ground state search computed only with simple update. We notice that even though the convergence of SU(2) is faster than of U(1) iPEPS in both cases, in the end, for $D = 7$, the energies computed with gradient-based optimization on a U(1) iPEPS are lower than the results obtained via simple update for both U(1) and SU(2) iPEPS. Here both ways are sufficient to obtain a very accurate solution, but again AD-based optimization achieves up to an order of magnitude better result, with a relative error $\mathcal{O}(10^{-4})$ for simple update, and $\mathcal{O}(10^{-5})$ for L-BFGS at $D = 7$.

CTMRG for this model needed no more than 10 iterations to converge, and L-BFGS required no more than 20 gradient and energy calculations per bond dimension. The performance of AD for this case was slightly better than for the Heisenberg model, with $\frac{T_f}{T_i} \sim 4 - 5$ after the checkpoints are applied.

3.2 The Hubbard model

The Hubbard model [Hub67] captures the essence of strongly correlated electron systems via the Hamiltonian:

$$H = -\sum_{i,j,\sigma} t_{ij} (c^\dagger_{i\sigma} c_{j\sigma} + c^\dagger_{j\sigma} c_{i\sigma}) + U \sum_i n_{i\uparrow} n_{i\downarrow} \tag{3.4}$$

t_{ij} are coupling amplitudes for the hopping from site i to site j and U simulates the Coulomb repulsion between fermions. In general, the ground state of the Hubbard model

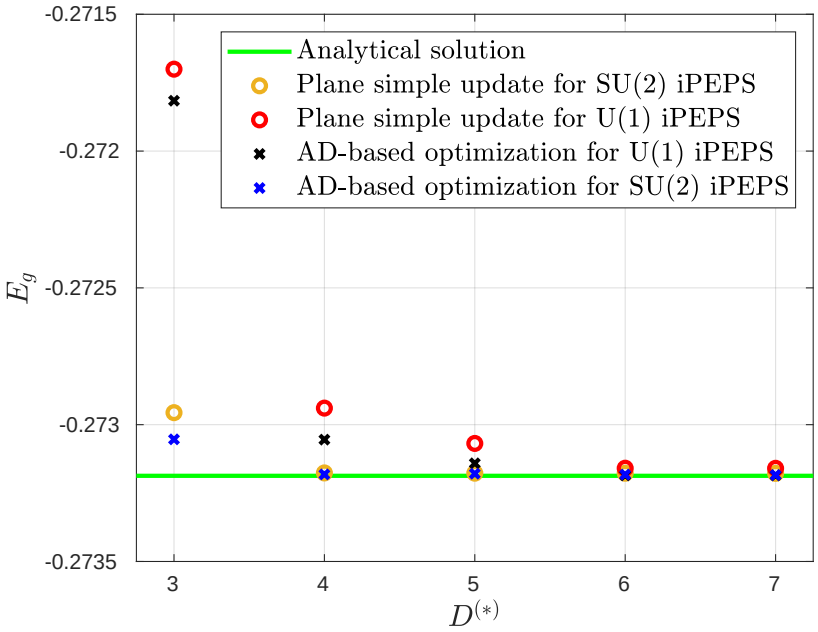


Figure 3.2: Variational energy for the 2D free-fermion model (3.2) on a square lattice as function of D, for U(1) symmetry, or D^* for SU(2). The data provided show the simple update results (circles) and the improved results obtained via L-BFGS optimization (x's), together with the analytical result (green line).

is characterized by a complicated phase diagram with rich physics and is yet to be fully understood. For this reason, it is important to have accurate simulations of the Hubbard model across a range of interaction parameters. Moreover, investigations of the Hubbard model using square-lattice iPEPS are significant because they allow for the exploration of high-T_c superconductivity in cuprates.

3.2.1 Nearest-neighbor interaction

Firstly we will focus on nearest-neighbor interactions, with a Hamiltonian

$$H = -\sum_{\langle i,j\rangle,\sigma} (c^\dagger_{i\sigma} c_{j\sigma} + c^\dagger_{j\sigma} c_{i\sigma}) + U \sum_i n_{i\uparrow} n_{i\downarrow} + \mu \sum_i n_i \tag{3.5}$$

for $U = 8$ and $\mu = -U/2$. The chemical potential μ enters as a parameter to control doping, as there is no direct way to do this with iPEPS. With this choice of μ, we ensure that each site is occupied by one fermion. With these parameters, the double occupancy is $\sim 10^{-2}$.

The choice of symmetry is again U(1) and the iPEPS supercell 2×2. In Fig. 3.3, the optimization landscape of L-BFGS is shown for bond dimensions $D = 2, 3, 4, 5, 6, 7, 8$. Note now that the variational energy is $E = E_g - \mu n$ where in this case $n = 1$ is the site

occupancy. What we observe is that even though for small D L-BFGS provides a large improvement over the corresponding simple update results, as the bond dimension increases it converges similarly to the simple update and therefore provides less and less advantage. This is a weak point in the protocol followed in this thesis, as, even though simple update provides a good initialization for the optimizations, the symmetry structure of the tensors throughout the whole optimization gets necessarily restricted by it. Additionally, we observe that a bigger advantage occurs due to the increase of bond dimension than due to the L-BFGS optimization.

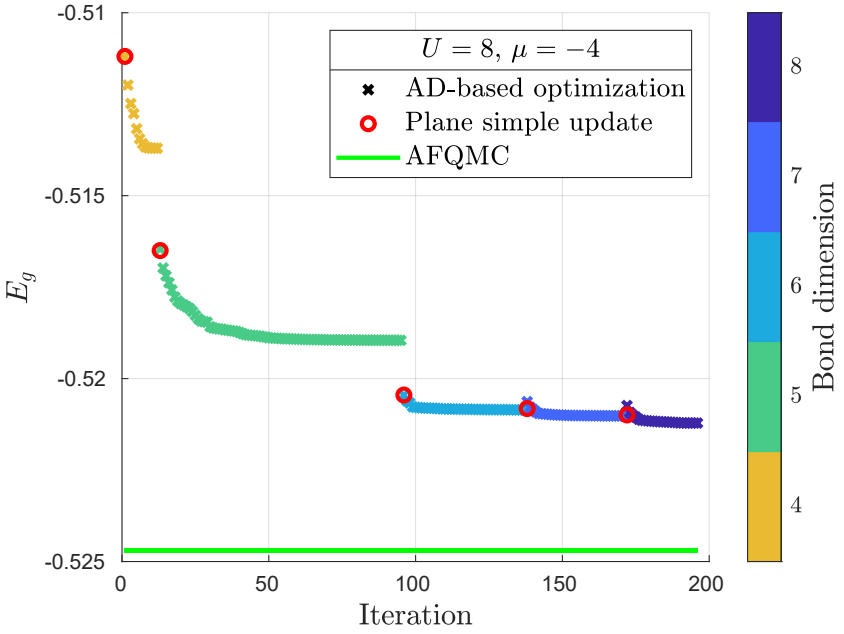


Figure 3.3: AD-based optimization for the NN Hubbard model at half-filling for a U(1) iPEPS and 2×2 supercell. The whole L-BFGS progression is plotted following the protocol described in Sec. 3.1. The color-bar indicates the bond dimension for which the similarly colored data were obtained. The results are compared to the simple update results(circles), as well as with an auxiliary-field quantum Monte Carlo (green) result [L+15]

.

Subsequently, the belief propagation gauging technique (Sec. (2.2)) was tested for the same model parameters. Firstly, the claims of the original publication [TF23] were tested to determine whether simple update enhanced with BP gauging converges to lower energies for ground-state iPEPS. For this, the optimal simple update result was picked for $D = 2$ and then, every time D was increased by one, simple update was carried out within the BP gauge in two ways: first, by converging the message tensors to the BP fixed-point after each gate update, and second, by skipping this step. It was found that BP has a very small impact on the ground-state energies, of the order of 10^{-5}, if simple update is well converged. With the application of L-BFGS optimization the impact of BP disappears, as

BP converges to tensors with the same symmetry structure as regular simple update. An example of this is given in Fig. 3.4 for $D = 4$, where the optimization landscape is given for both cases.

Note also that performing variational optimization via optimizing energy 2.4 under the improved BP framework cannot provide better results since BP already gives the best approximation of rank-one environment that we can have, and from there one has to resort to more accurate methods such as CTMRG.

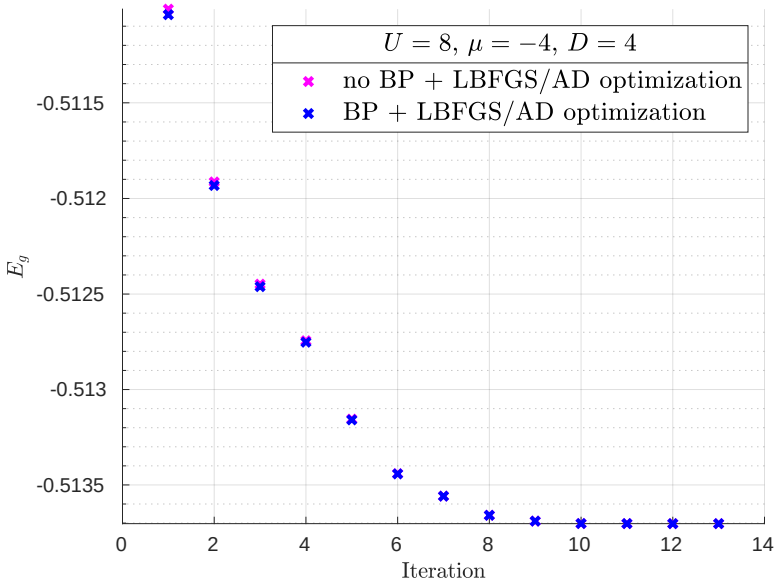


Figure 3.4: Optimization landscape of variational energy for the NN Hubbard model at half-filling, with a U(1) iPEPS with 2×2 supercell and $D = 4$. The advantage of converging the message tensors to the fixed point at each gate update of simple update emerges at the initial state as a slightly decreased energy, but eventually gets lost as L-BFGS progresses.

3.2.2 Next-nearest-neighbor interaction

Finally, we study the next-nearest neighbor Hubbard model with Hamiltonian:

$$H = -\sum_{\langle i,j\rangle,\sigma}(c^\dagger_{i\sigma}c_{j\sigma} + c^\dagger_{j\sigma}c_{i\sigma}) - t'\sum_{\langle\langle i,j\rangle\rangle,\sigma}(c^\dagger_{i\sigma}c_{j\sigma} + c^\dagger_{j\sigma}c_{i\sigma}) + U\sum_i n_{i\uparrow}n_{i\downarrow} + \mu\sum_i n_i \qquad (3.6)$$

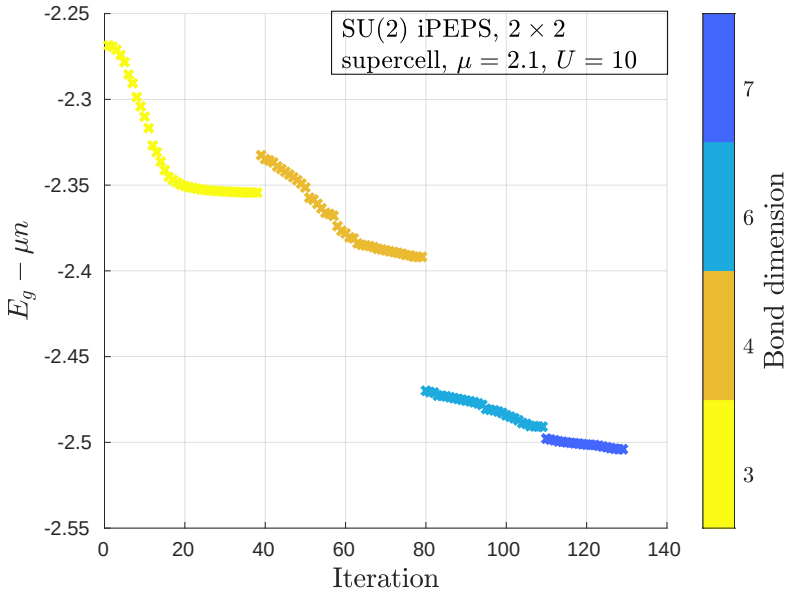
The next-nearest neighbor coupling t' introduces frustration to the system, and for this reason the model is challenging to study as it incorporates different phases. An interesting aspect is the work on the disambiguation on the nature of the ground state, as different simulations give inconsistent results. In particular, some simulations indicate an antiferromagnetic striped ground state with a periodicity of 8 for the spin order and 4 for the

charge order [PCC19], while others indicate a uniform ground state which exhibits d-wave superconductivity. In [ZLvD23], for $t' = 0.25$ and for SU(2) symmetric iPEPS, ground state energies were shown to be lower than for U(1) symmetric iPEPS at large doping, with simulations using simple update up to $D = 12$.

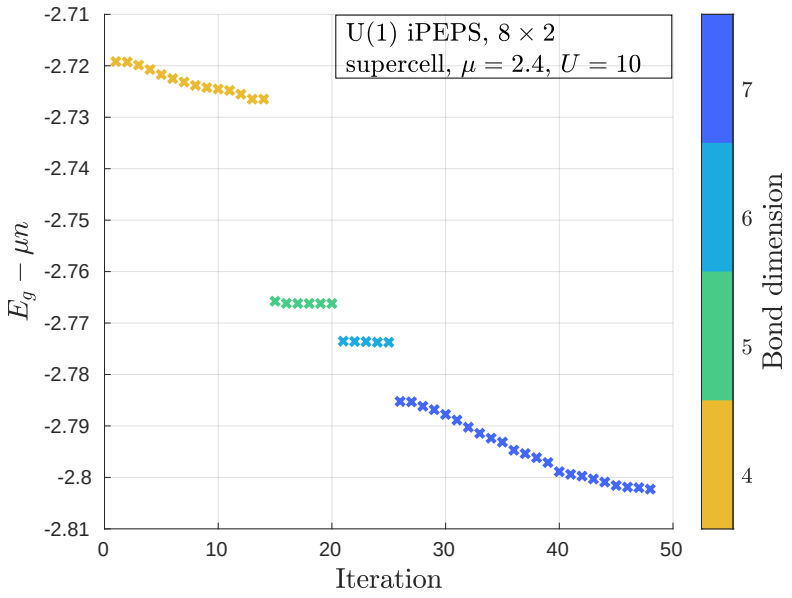
In Fig. 3.5 we show the AD-optimized results of a U(1) symmetric iPEPS and a SU(2) symmetric one, for interaction strength $U = 10$ and NNN hopping amplitude $t' = 0.25$. The variational energy $E_g - \mu n$ is plotted with the number of iterations of L-BFGS for bond dimensions $D = 3, 4, 6, 7$ and $D = 4, 5, 6, 7$ for SU(2) and U(1) symmetric iPEPS accordingly. In the former case, $\mu = 2.1$ is chosen, and in the latter $\mu = 2.4$. For the SU(2) symmetric iPEPS we observe a similar pattern as in Fig. 3.3, namely that L-BFGS provides a significant improvement on the variational energy for small bond dimensions, and we also notice that it converges as the bond dimension increases. For the U(1) iPEPS we note that convergence seems to be slower, as for the $D = 7$ there is still significant decrease to the variational energy. The fact that there are more variational parameters than the SU(2) might play a role in this. A bond dimension higher than $D = 7$ was not achieved in the limited time of this project, due to the increased computation times. Results for higher bond dimensions might result in a better convergence of the optimization landscape, however for the large 8×2 supercell it is challenging to reach high bond dimensions.

Some technical differences compared to the previous examples should be noted here. Firstly, the initialization of these two iPEPS was not obtained via the simple update algorithm. The converged state for $D = 12$ simple update was already known, and therefore the corresponding results for smaller bond dimensions were obtained via a projection from the $D = 12$ states.

Secondly, variational optimization on the NNN Hubbard model proved more challenging compared to the previous, simpler models. Specifically, optimizing the quantity $E_g - \mu n$ with the usual L-BFGS trial step $a_0^{(k)} = 1$ had the tendency to not converge to the ground state of a specific phase, but to optimize over the term μn of the variational energy, instead of E_g which is what we desire, and as a result to slip between different phases with different ground state energies. For this reason, a small step $a_0^{(k)} = 0.1$ was chosen as a trial step of the line search of L-BFGS, at the cost of a slower, less-than-quadratic convergence.



(a)



(b)

Figure 3.5: AD-based optimization of the NNN doped Hubbard model. (a) Variational landscape of L-BFGS for SU(2) iPEPS with a 2×2 supercell. (b) Variational landscape of L-BFGS for U(1) iPEPS with an 8×2 supercell.

Outlook

In this thesis, reverse-mode Automatic Differentiation (AD) was implemented for the QSpace library and applied to the Corner Transfer Matrix Renormalization Group, with the scope of ground-state search of tensor networks at the thermodynamic limit. We saw that even though the basic principle of AD is simple, there are several technical details which must be overcome in order to achieve a stable and efficient implementation.

Gradient-based optimization via AD has several advantages. It is a very flexible algorithm, which can be implemented to any tensor network algorithm with only few modifications to it. Gradient calculations require a time cost of the same order of magnitude as that of the forward sweep, and when combined with a quadratically-converging optimization and a good initial guess for the ground-state, it only needs a few tens of iterations to converge to a solution for a fixed bond dimension. Additionally, if initialized with the ground state obtained via the simple update algorithm, we can gain insight over the performance of simple update for tensor networks with loops, and over the variational landscape around simple-update solutions. Overall, limited-memory BFGS proved to be a useful method to improve the ground state energies obtained via simple update for a fixed bond dimension. It is therefore reasonable to apply it directly at a target bond dimension D, and initialize it with the optimal simple-update ground state tensors with bond dimension D.

Yet, we saw that there are challenges in the implementation, which we need to overcome in order to acquire competitive results. Firstly, as the bond dimension increases, it becomes computationally expensive to cope with the tens of iterations needed to optimize the tensors. On top of that, we saw that with increasing bond dimension the improvement which gradient information provides to the variational energy becomes less advantageous as compared to the much cheaper alternative, the energy obtained via simple update. This is expected as the optimization will eventually converge, however the fact that it converges so closely to the simple update solutions (for large D) implies that it might be limited by the initialization via the simple update ground state. This drawback might be overcome by modifying the Limited-memory BFGS algorithm in order to explore different symmetry sectors.

Appendix A

Gauging of loop-free tensor networks

Loop-free tensor networks are special because entanglement can be encoded in bonds between connected tensors. For this reason they are easier to deal with and tensor network methods are particularly successful for such geometries. The most basic example is that of a Matrix Product State (MPS), made up of degree-three tensors with two virtual bonds contracted to neighboring tensors and one physical bond corresponding to one particle per site. This can be represented with the following diagram:

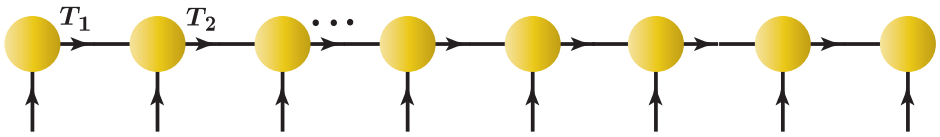


Figure A.1: Example of an MPS.

Such a diagram, as all the tensor networks with tree-like connections, has the property that cutting a bond splits the network in two. This is a particularly important property as it means that entanglement between the left and right cut can be described by the Schmidt coefficients. In this way a Schmidt decomposition at the first bond:

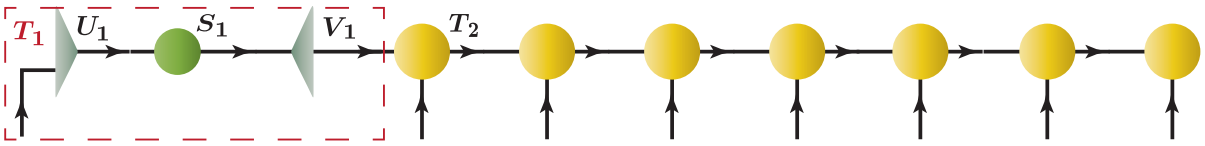


Figure A.2: Through an SVD at the first site $T_1 = U_1 S_1 V_1$ the entanglement of the first site with the rest is encoded in matrix S_1.

results in a bond tensor S_1, which encodes entanglement between the first site and the rest. Entanglement entropy in an MPS has a tendency to drop quickly with the distance. Due to this property, truncation of the smallest singular values can decrease the dimensions of

the tensors without altering significantly the state, making the computations more stable and efficient.

These properties of loop-free tensor networks can be used to obtain two very important gauges, the canonical form and the $\Gamma - \Lambda$ form. They can also be generalized for infinite loop-free tensor networks. These gauges are described in this appendix, in connection mainly to the discussion of Sec. 2.2.

A.1 Canonical form

For the *left canonical form*, starting from an MPS as that of Fig. A.1, we use an SVD at the first site to obtain the form of Fig. A.2. Then, the steps in Fig. A.3 bring the MPS to a form made up of left isometries. The isometric property of the left isometries $U_j^\dagger U_j = \mathbb{I}$ is diagrammatically described by Fig. A.4. A similar *right canonical form* can be obtained via an analogous procedure starting from right to left, in which the final configuration is a decomposition of right isometries with the property $V_j V_j^\dagger = \mathbb{I}$.

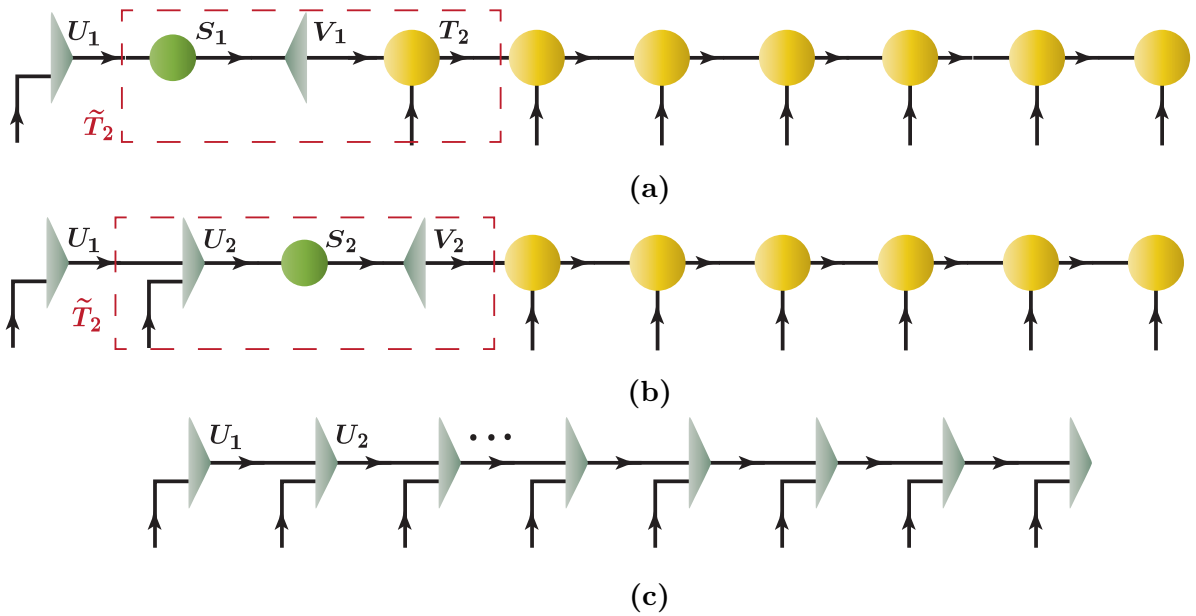


Figure A.3: (a) Starting from the configuration of Fig. A.2 the tensors $S_1 V_1$ get absorbed to tensor T_2, defining a new tensor $\tilde{T}_2$. (b) Decompose tensor $\tilde{T}_2$ through an SVD. (c) The procedure is continued until all the tensors have the form of left isometries.

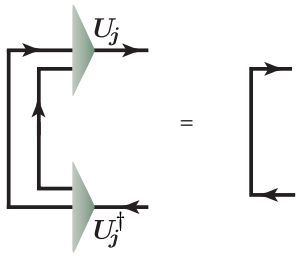


Figure A.4: Diagrammatic representation of isometry $U_j^\dagger U_j = \mathbb{I}$.

A.2 $\Gamma - \Lambda$ form

The $\Gamma - \Lambda$ form is a gauge made up of site tensors Γ_j and bond tensors Λ_j as shown in Fig. A.5. The $\Gamma - \Lambda$ form is closely related to the left and right canonical forms through:

$$\begin{aligned} U_j &= \Lambda_{j-1}\Gamma_j, \quad l \geq j \geq 1 \\ V_j &= \Gamma_j \Lambda_{j+1}, \quad L \geq j > l \end{aligned} \tag{A.1}$$

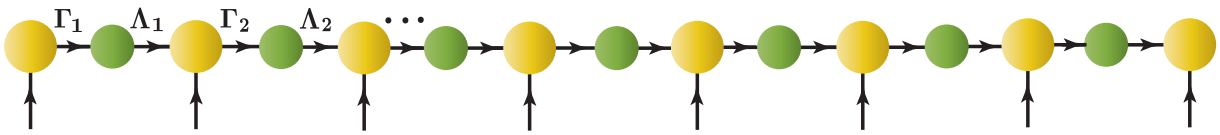
where L is the number of sites and l is a target site (for which potentially we would like to obtain a local expectation value). The $\Gamma - \Lambda$ form can be obtained from the equations:

$$\begin{aligned} \Lambda_0 &= \Lambda_L = 1 \\ \Lambda_j &= \begin{cases} S_j^{(left)}, & l \geq j \geq 1 \\ S_j^{(right)}, & L \geq j > l \end{cases} \\ \Gamma_j &= \begin{cases} \left(S_{j-1}^{(left)}\right)^{-1} U_j, & l \geq j \geq 1 \\ V_j \left(S_{j+1}^{(right)}\right)^{-1}, & L \geq j > l \end{cases} \end{aligned} \tag{A.2}$$

where $S_j^{(left/right)}$ are the singular-value matrices obtained when bringing the MPS to left/right canonical form up to site $l/l+1$.

In this gauge, for each site the left/right bond tensors encode the entanglement from the left/right part of the MPS. This gauge becomes approximative in practice, when truncation of the very small singular values is applied to the SVDs and at the inversion of S_j.

Due to Eq. (A.1) and Fig. A.4, as well as the analogous equation for right isometries, the $\Gamma - \Lambda$ gauge also respects isometric conditions as depicted in Fig. A.6, which are approximative when truncations are involved. These conditions assure that the environment for each site is captured accurately by the bond tensors. This can be seen when calculating local observables. Fig. A.7 shows that the expectation value of a 2-site observable $O_{l,l+1}$ can be obtained only by tensors $\{\Lambda_{l-1}, \Gamma_l, \Lambda_l, \Gamma_{l+1}, \Lambda_{l+1}\}$ using identities A.6.



(a)

Figure A.5: The $\Gamma - \Lambda$ gauge for an MPS.

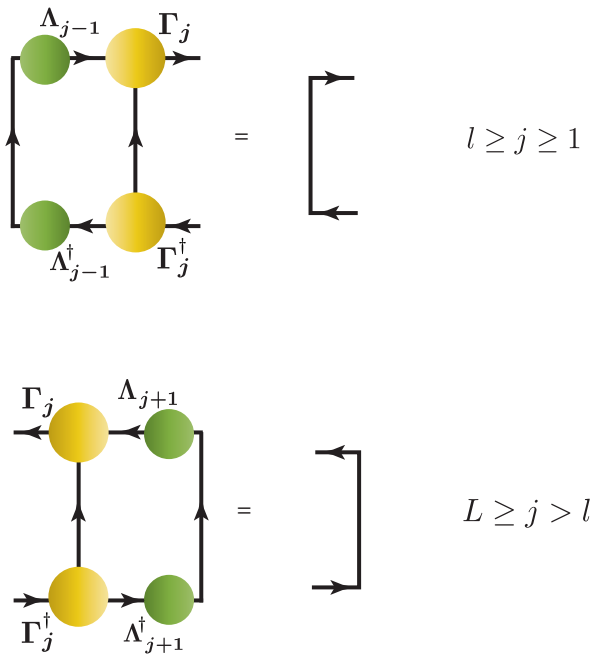


Figure A.6: Isometric condition of the $\Gamma - \Lambda$ gauge.

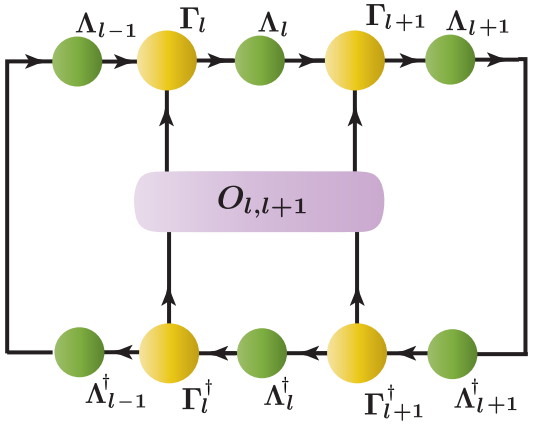


Figure A.7: Expectation value of a two-site observable $O_{l,l+1}$ for an MPS at the $\Gamma - \Lambda$ gauge.

Appendix B

Tensor derivative results for reverse-mode automatic differentiation

In this appendix there are collected a few results for reverse-mode AD of operations with tensors which are used commonly in tensor networks, together with some technical details to make the implementation explicit.

General contraction

The most used operations in tensor networks are contractions, and so it is important to present the formula for its gradient for a general case. Let's start from a general contraction which takes place during the forward sweep of the algorithmic procedure:

$$C^{c_1 c_2 \dots c_l}{}_{c_{l+1} \dots c_{d_C}} = A \circ_{idxA} B = \sum_{\tilde{a} \in idxA} A^{a_1 \dots a_m}{}_{a^*_{m+1} \dots a^*_{d_A}} B^{b_1 \dots b_n}{}_{b^*_{n+1} \dots b^*_{d_B}} \tag{B.1}$$

where $d_{A/B}$ is the degree of tensors A/B, $idxA$ is the set of all indices of A that get contracted with B and the symbol $\circ_{idxA}$ is used to denote this operation. The set of indices of B which are contracted to indices in $idxA$ is denoted by $idxA^*$[1]. Suppose that $\overline{C}$ is known and we want to calculate $\overline{A}$. We need to consider two cases individually.

Case 1: $|idxA| < d_B$

In this case the indices of tensor C are

$$idxC = idxC_A \cup idxC_B \tag{B.2}$$

where

$$idxC_A = \{a_1^{(*)}, a_2^{(*)}, ..., a_{d_A}^{(*)}\} \setminus idxA \tag{B.3}$$

[1]This notation is chosen due to details on the conventions used in the QSpace library [Wei24]. The important is that if $i \in idxA$ is covariant, $i^* \in idxA^*$ is contravariant and vice versa.

and

$$idxC_B = \{b_1^{(*)}, b_2^{(*)}, ..., b_{d_B}^{(*)}\} \setminus idxA^* \tag{B.4}$$

where $i^{(*)}$ is used to specify that the corresponding index can be either contravariant or covariant[2]. The corresponding indices of $\overline{C}$ are those of C but starred, and are denoted by $idxC^*$. This can be understood by noticing that in the diagrammatic representation of gradient tensors (Fig. 1.6), an inward arrow in C would be an outward arrow in $\overline{C}$. Then, up to permutations of the final indices the following equation holds:

$$\overline{A}_{a_1^*...a_m^*}{}^{a_{m+1}...a_{d_A}} = \overline{C} \circ_{idxC_B^*} B \tag{B.5}$$

The convention used is that $\overline{A}$ should have the same order of indices as A (starred), so a permutation may be needed afterward.

Example Let's consider the contraction:

$$C^{a_1a_3}{}_{a_4^*a_5^*}{}^{b_3} = \sum_{\tilde{a} \in idxA} A^{a_1a_2a_3}{}_{a_4^*a_5^*a_6^*a_7^*} B^{a_6a_7b_3}{}_{a_2^*} \tag{B.6}$$

represented in Fig. B.1a.

In this case $idxA = \{a_2, a_6^*, a_7^*\}$, $idxA^* = \{b_1 = a_6, b_2 = a_7, b_4^* = a_2^*\}$, $idxC_A = \{a_1, a_3, a_4^*, a_5^*\}$, $idxC_B = \{b_3\}$. Then,

$$\overline{A}_{a_1^*a_3^*}{}^{a_4a_5a_6a_7}{}_{a_2^*} = \sum_{b^{3*}} \overline{C}_{a_1^*a_3^*}{}^{a_4a_5}{}_{b_3^*} B^{a_6a_7b_3}{}_{a_2^*} \tag{B.7}$$

A permutation (1 7 2 3 4 5 6) is needed to get the correct indices for $\overline{A}$. The calculation of $\overline{B}$ would be analogous.

Case 2: $|idxA| = d_B$

Suppose now that in the previouse example also index b_3 is contracted and so $|idxA| = d_B$ and $idxC_B = \{\}$: Now we are dealing with a contraction of the form (Fig. B.2a):

$$C^{a_1a_3}{}_{a_4^*} = \sum_{\tilde{a} \in idxA} A^{a_1a_2a_3}{}_{a_4^*a_5^*a_6^*a_7^*} B^{a_5a_6a_7}{}_{a_2^*} \tag{B.8}$$

As can be seen in the Fig. B.2b, now a tensor product is needed to get $\overline{A}$ (up to permutations):

$$\overline{A} = \overline{C} \otimes B \tag{B.9}$$

[2]Even though distringuishing covariant from contravariant indices is not especially relevant for the cases considered, up and down indices are used to represent flow of charges as explained in Sec. 1.2.

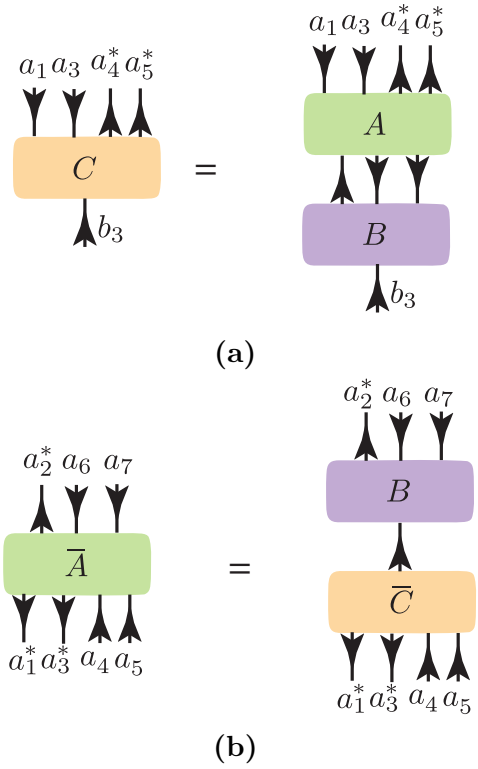


Figure B.1: (a) Diagrammatic representation of contraction (B.6). (b) Gradient with respect to tensor A gien by Eq. (B.7).

Singular value decomposition (SVD)

The backpropagation formula for the gradient of the SVD given by the equation

$$A = USV^\dagger \tag{B.10}$$

is [WZ19]:

$$\overline{A} = \left[U\overline{S}V^\dagger + UPV^\dagger + \left(I - UU^\dagger\right)\overline{U}S^{-1}V^\dagger + US^{-1}\overline{V}^{\dagger *}\left(I - VV^\dagger\right)\right]^* \tag{B.11}$$

where

$$\begin{aligned} P &= M + N \\ M &= \tfrac{1}{2}\left(F - G - S^{-1}\right) \odot \left(V^\dagger \overline{V}^{\dagger T} - (V^\dagger \overline{V}^{\dagger T})^\dagger\right) \\ N &= \tfrac{1}{2}\left(F + G\right) \odot \left(U^T \overline{U} - (U^T \overline{U})^\dagger\right), \end{aligned} \tag{B.12}$$

$\odot$ denotes the element-wise multiplication, and

$$\begin{aligned} S_{ii} &= s_i \\ F_{ij} &= \tfrac{1}{s_j - s_i} \\ G_{ij} &= \tfrac{1}{s_i + s_j} \end{aligned} \tag{B.13}$$

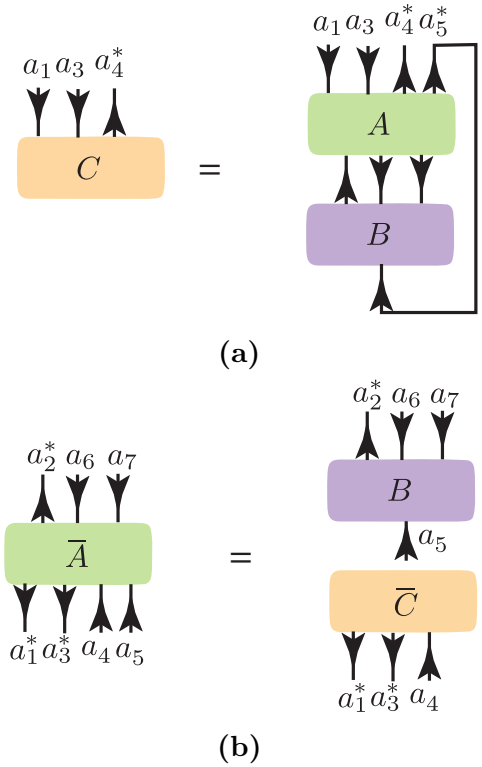


Figure B.2: (a) Diagrammatic representation of contraction (B.8). (b) Gradient tensor with respect to A. At the right hand side no contraction is depicted, which implies that a tensor product should be taken to map to the space where A belongs.

Note that the notation has been changed compared to the reference [WZ19] in order to match the notation and conventions of this thesis.

In practice, to avoid instabilities for very small singular values, F is modified as:

$$F_{ij} = \frac{s_j - s_i}{(s_j - s_i)^2 + \epsilon} \tag{B.14}$$

for some small $\epsilon \sim 10^{-11} - 10^{-12}$, in order to keep the accuracy to machine precision. For the same reason we use the following G:

$$G_{ij} = \begin{cases} \frac{1}{s_j + s_i} & s_j + s_i > \epsilon \\ 0 & s_j + s_i \leq \epsilon \end{cases} \tag{B.15}$$

Fixed-point iteration

Another operation that is used often in tensor networks is a fixed-point condition of the form:

$$y^* = f(y^*, u) \tag{B.16}$$

where u are parameters with respect to which the gradient $\nabla_u E$ of some larger function E must be obtained (E is for example the expectation value of the Hamiltonian). As it turns out, obtaining $\bar{u}$ can be transformed to a fixed-point condition for the gradient tensors, and the graph of only the last iteration (i.e. when Eq. (B.16) holds) can be retained [Chr94]. A summary of this process is described below.

Differentiating Eq. (B.16) with respect to u we get:

$$\bar{u} = \overline{y^*}\frac{\partial y^*}{\partial u} = \overline{y^*}\left(\mathbb{I} - \frac{\partial f(y^*, u)}{\partial y^*}\right)^{-1}\frac{\partial f(y^*, u)}{\partial u} \tag{B.17}$$

Then, expanding the parenthesis in a geometric series we have:

$$\bar{u} = \overline{y^*}\left(\mathbb{I} + \frac{\partial f(y^*, u)}{\partial y^*} + \left(\frac{\partial f(y^*, u)}{\partial y^*}\right)^2 + ...\right)\frac{\partial f(y^*, u)}{\partial u} \tag{B.18}$$

In order to approximate Eq. (B.18) in a real implementation, we retain the graph of the last iteration, which is:

$$y_f = f(y_i, u) \tag{B.19}$$

where y_i are the initial tensors of the last iteration and y_f the final ones. If the forward sweep is well converged, $y_f \approx y_i \approx y^*$. The first two terms of Eq (B.18) can be obtained via Eq. (B.19):

$$\bar{u} = \overline{y_f}\frac{\partial f}{\partial u} + \overline{y_i}\frac{\partial y_i}{\partial u} \tag{B.20}$$

The whole Eq. (B.18) can be derived via subsequent reverse sweeps over the same f (at the fixed-point), by updating $\overline{y_f}$ as:

$$\overline{y_f} \to \overline{y_f} + \overline{y_i} \tag{B.21}$$

and updating $\overline{y_i}$ via backpropagation, starting from this new $\overline{y_f}$. In this way we can reach a fixed-point for $\overline{y_f}$. Each time, the $\bar{u}$ and $\overline{y_i}$ must be re-initiated to 0 so that no false accumulation of gradients happens.

This method could be applied to a CTMRG algorithm with the memory consumption of only one iteration. However, due to the gauge freedom of tensor networks, Eq. B.16 is not guaranteed to hold.

Bibliography

[Bau74] F. L. Bauer. Computational graphs and rounding error. *SIAM J. Numerl. Anal.*, 11:87–96, 1974.

[BBCD00] M. Bahrtholomew Biggs, S. Brown, B. Christianson, and L. Dixon. Automatic differentiation of algorithms. *J. Comput. Appl. Math.*, 124:171–190, 2000.

[CE06] M. Cramer and J. Eisert. Correlations, spectral gap and entanglement in harmonic quantum systems on generic lattices. *New J. Phys.*, 8(71), 2006.

[CEP07] M. Cramer, J. Eisert, and M. B. Plenio. Statistics dependence of the entanglement entropy. *Phys. Rev. Lett.*, 98:220603, 2007.

[Chr94] B. Christianson. Reverse accumulation and attractive fixed points optim. methods software. *Optim. Methods Software*, 3:311–326, 1994.

[COBV10] 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.

[CPGSV21] I. J. Cirac, D. Pérez-García, N. Schuch, and F. Verstraete. Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Rev. Mod. Phys.*, 93:045003, 2021.

[CV09] P. Corboz and G. Vidal. Fermionic multiscale entanglement renormalization ansatz. *Phys. Rev. B*, 80:165129, 2009.

[ECP10] J. Eisert, M. Cramer, and M. B. Plenio. Colloquium: Area laws for the entanglement entropy. *Rev. Mod. Phys.*, 82:277–306, 2010.

[FR64] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Comp. J.*, 7(2):149–154, 1964.

[GK06] D. Gioev and I. Klich. Entanglement entropy of fermions in any dimension and the widom conjecture. *Phys. Rev. Lett.*, 96:100503, 2006.

[Gri89] A. Griewank. *On Automatic Differentiation, Math. Program. Recent Dev. Appl.*, pages 83–108. Kluwer Academic Publishers, 1989.

[GW08] A. Griewank and A. Walther. *Evaluating derivatives: Principles and techniques of automatic differentiation*, pages 19–29. Society for Industrial and Applied Mathematics, Philadelphia, 2nd edition, 2008.

[Has07] M. B. Hastings. An area law for one-dimensional quantum systems. *J. Stat. Mech.*, 2007.

[Hub67] J. Hubbard. Electron correlations in narrow energy bands V. A perturbation expansion about the atomic limit. *Proc. R. Soc. London A*, 296(82), 1967.

[JOV+08] 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.

[JWX08] 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.

[L+15] J. P. F. LeBlanc et al. Solutions of the two-dimensional Hubbard model: Benchmarks and results from a wide range of numerical algorithms. *Phys. Rev. X*, 5:041041, 2015.

[LLWX19] H.-J. Liao, J.-G. Liu, L. Wang, and T. Xiang. Differentiable programming tensor networks. *Phys. Rev. X*, 9:031041, 2019.

[LS23] I. V. Lukin and A. G. Sotnikov. Variational optimization of tensor-network states with 1159 the honeycomb-lattice corner transfer matrix,. *Phys. Rev. B*, 107:054424, 2023.

[NW99] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research, 1st edition, 1999.

[NWR+24] J. Naumann, E. L. Weerda, M. Rizzi, J. Eisert, and P. Schmoll. An introduction to infinite projected entangled-pair state methods for variational ground state simulations using automatic differentiation. *SciPost Phys. Lect. Notes*, 86, 2024.

[OV09] R. Orús and G. Vidal. Simulation of two-dimensional quantum systems on an infinite lattice revisited: Corner transfer matrix for tensor contraction. *Phys. Rev. B*, 80:094403, 2009.

[PCC19] B. Ponsioen, S. S. Chung, and P. Corboz. Period 4 stripe in the extended two-dimensional Hubbard model. *Phys. Rev. B*, 100:195141, 2019.

[PEDC05] M.B. Plenio, J. Eisert, J. Dreißig, and M. Cramer. Entropy, entanglement, and area: Analytical results for harmonic lattice systems. *Phys. Rev. Lett.*, 94:060503, 2005.

[San97] A. W. Sandvik. Finite-size scaling of the ground-state parameters of the two-dimensional Heisenberg model. *Phys. Rev. B*, 56(18), 1997.

[Spe64] M. R. Speigel. *Schaum's outline of theory and problems of complex variables with an introduction to conformal mapping and its applications*, page 69. McGraw Hill, Inc., 1964.

[SvBdL12] L. Sorber, M. van Barel, and L. de Lathauwer. Unconstrained optimization of real functions in complex variables. *SIAM J. Optim.*, 22(3):879–898, 2012.

[TF23] J. Tindall and M. Fishman. Gauging tensor networks with belief propagation. *SciPost Phys.*, 15:222, 2023.

[TW05] M. Troyer and U. Wiese. Computational complexity and fundamental limitations to fermionic quantum Monte Carlo simulations. *Phys. Rev. Lett.*, 94:170201, 2005.

[VC04] F. Verstraete and J. I. Cirac. Renormalization algorithms for quantum-many body systems in two and higher dimensions. arXiv:cond-mat/0407066v1[cond-mat.str-el], 2004.

[vdB94] A. van den Bos. Complex gradient and hessian. *IEE Proc. Vis. Image & Signal Process.*, 141(6):380–382, 1994.

[Vid07] G. Vidal. Classical simulation of infinite-size quantum lattice systems in one spatial dimension. *Phys. Rev. Lett.*, 98:070201, 2007.

[Wei24] A. Weichselbaum. Qspace – an open-source tensor library for abelian and non-abelian symmetries. arXiv:2405.06632 [cond-mat.str-el], 2024.

[Wir27] W. Wirtinger. Zur formalen Theorie der Funktionen von mehr komplexen Veränderlichen. *Math. Ann.*, 97:357–375, 1927.

[Wol06] M. M. Wolf. Violation of the entropic area law for fermions. *Phys. Rev. Lett.*, 96:010404, 2006.

[WR24] E. L. Weerda and M. Rizzi. Fractional quantum hall states with variational projected entangled-pair states: A study of the bosonic harper-hofstadter model. *Phys. Rev. B*, 109:L241117, 2024.

[WZ19] Z.-Q. Wan and S.-X. Zhang. Automatic differentiation for complex valued SVD. arXiv:1909.02659[math.NA], 2019.

[Zha21] C. Zhang. Symmetric infinite projected entangled-pair state study of quantum lattice models. Master's thesis, Ludwig–Maximilians–Universität München, 2021.

[ZLvD23] C. Zhang, J.-W. Li, and J. von Delft. Frustration-induced superconductivity in the $t - t'$ hubbard model. arXiv:2307.14835v1 [cond-mat.str-el], 2023.

[ZP20] M. P. Zaletel and F. Pollmann. Isometric tensor network states in two dimensions. *Phys. Rev. Lett.*, 124:037201, 2020.