Full MERA Network Optimization Tool

Cole A. Coughlin[*]
Department of Physics, University of Winnipeg

Abstract
In recent years, tensor networks have proven to be a useful tool for simulating certain quantum systems, quantum error correction, condensed matter physics, and some networks such as the Multi-scale Entanglement Renormalization Ansatz (MERA) have some holographic properties that are of interest to researchers in quantum gravity. The program that this report describes is designed to optimize a MERA network that simulates a quantum system with a transverse Ising Hamiltonian. Other implementations make use of symmetries within the network like scale invariance and translation invariance, but this code was developed strictly to simulate the entire network to then be able to modify any tensors in the network and have full control of the system.

1 Introduction

The Multi-scale Entanglement Renormalization Ansatz (MERA) is a tensor network that can be used to accurately and quickly approximate ground state energies of quantum lattices. [1] This software is designed to simulate a 1 dimensional quantum lattice with periodic boundary conditions which in some sense can be thought of existing on the outermost layer of the MERA network. This code allows the user to create a MERA object by specifying the number of layers of the network, the number of branches from the top tensor, and the number of top tensors of the network (under construction).

2 MERA Class

2.1 Variables and stored parameters

The MERA object initializes the following variables outside of the constructor

- h - the transverse magnetic field applied to the spins
- sX, sY, sZ - the Pauli matrices
- ham_s - helper matrix used when initializing Hamiltonian
- ham_init - initial Hamiltonian for Ising model
- chi - bond dimension exiting isometries from above
- chi_p - bond dimension exiting disentanglers from above
- chi_b - bond dimension on finest layer for the actual spin states

and the constructor for the MERA initializes the following parameters when the object is created

- branches - this is the number of initial "branches" from the top tensor
- layers - the number of disentangler and isometry layers in the network
- numTopTensors - the number of top tensors
- uDis - array in which the disentangler tensors are stored

[*]E-mail: coughlic@myumanitoba.ca

– wIso - array in which the isometry tensors are stored
– rhoThree - array in which the upper enviroment tensors are stored
– hamThree - array in which the Hamiltonian or effective Hamiltonian tensors are stored
– n_sites - the number of spins on the one dimensional spin lattice

2.2 Constructor method

The constructor for the MERA object is passed the number of branches from the top tensor, the number of layers in the network, and the number of top tensors. The constructor then calculates the number of sites by multiplying the number of branches by 2 to the power of the number of layers, since the number of outgoing indices doubles with every layer. It then calculates the total number of isometries and disentanglers so it can initialize the arrays of tensors. The constructor then initializes every tensor in the network. All isometries and disentanglers begin as randomized tensors and are initialized according to the bond dimension necessary for the layer. All of the Hamiltonians on the outermost layer are initialized according to the Ising model Hamiltonian of the system and the rest are initialized randomly.

2.2.1 Bond Dimension

The following diagram shows the bond dimension of different tensors. The only difference between the outtermost layer and every other layer is that chi_p is replaced with chi_b in the last layer.

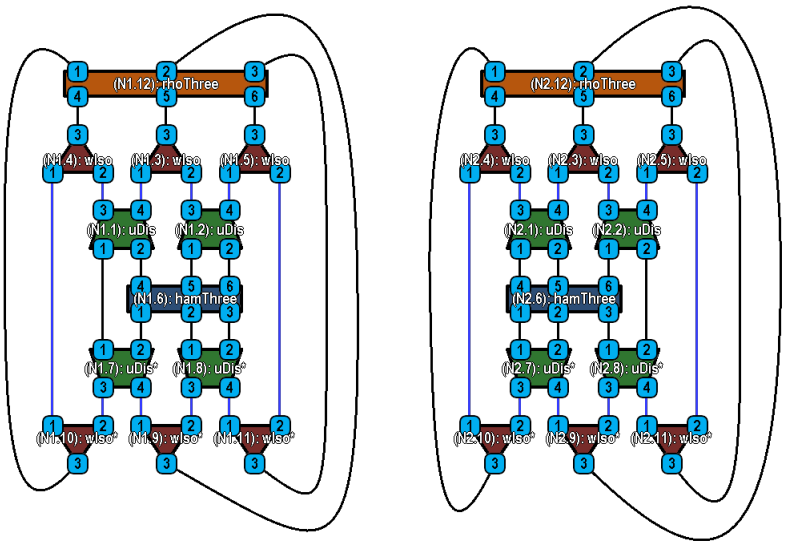

Figure 1: Tensor network used to optimize all tensors in the network. Black indices have bond dimension chi, blue indices have either chi_b or chi_p depending on the layer. Diagram and python file binaryMERAfull3.py were created with TensorTrace software

2.3 Helper Methods

2.3.1 getLoweredRho(leadingEdge, layer)

This method receives the index of the leading edge around the current layer that is being optimized and the current layer and returns the index of the rhoThree that will be updated when lowering the rhoThree of the current position.

2.3.2 getLiftedHamiltonian(leadingEdge, layer)

This method receives the index of the leading edge around the current layer that is being optimized and the current layer and returns the index of the hamThree that will be updated when lifting the Hamiltonian of the current position.

2.3.3 getEnviroment(leadingEdge, layer)

This method receives the index of the leading edge around the current layer that is being optimized and the current layer and returns all of the tensors needed to perform calculations with the tensor networks in figure 1 and in the current position. If the layer being optimized is the top layer, then the top tensor must be transposed based on the position around the network relative to the top tensor, since the numbering of the indices of the rhoThree being updated will need to be changed at different positions.

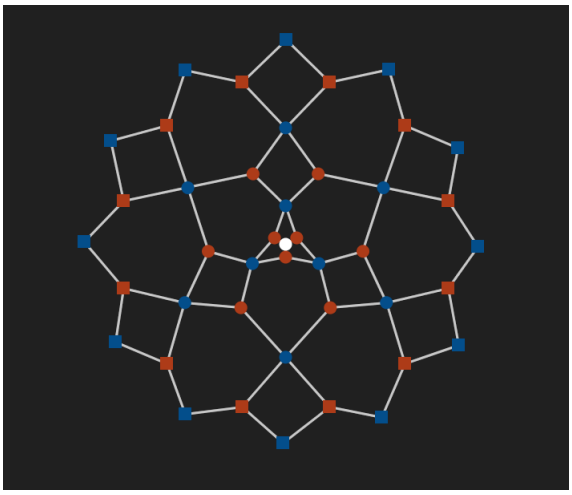

Figure 2: Typical MERA network created by the software. Meant to illustrate that the top tensor will be shared by different parts of the network and this needs to be taken into account when performing calculations on the top layer

2.4 Optimization of network

The optimization algorithm of the network follows closely the example for the optimization of a scale invariant MERA on the TensorTrace software website, but instead of updating each isometry and disentangler only once on each layer and treating them all as exact copies on each layer, every tensor in the network is updated individually. The algorithm sweeps over every tensor of each layer starting with the outtermost layer and working its way up the network. Once the top layer is optimized the top tensor is diagonalized and then propagated back down the network through each rhoThree of every layer. The average energy per site is then calculated by summing over the energy at each site and dividing by the number of sites. The optimization is repeated until the number of sweeps specified is reached.

3 Results

This implementation allows the efficient estimation of the ground state energies of quantum spin lattices in one dimension with very few sweeps through the network. A sample calculation of the ground state energy is shown below.

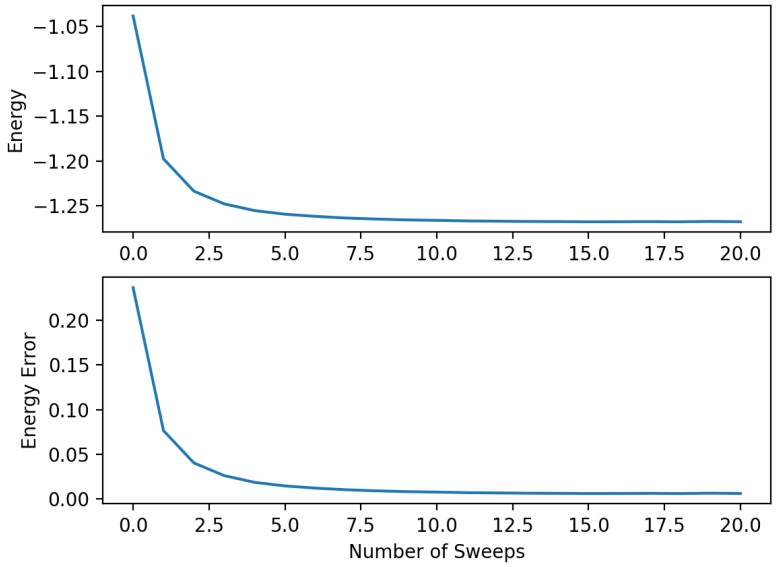

Figure 3: Convergence of the average ground state energy of a spin Ising model with a h = 1. The energy error compared to the exact energy of the system is also shown converging with the ground state energy.

4 Future Work

This software was developed in hopes to explore the holographic properties of MERA networks. Undoubtedly more functionality will be added to calculate other observables of the network like the entanglement entropy and correlation functions. Currently the number of top tensors in the network is not able to be changed from 1, but if deemed useful this functionality will also be added in the future. Other types of tensor networks may also be added in order to compare properties with MERA networks and their holographic properties.

5 Conclusion

Previous studies have provided methods for optimizing and performing calculations with translationally invariant MERA networks. While efficient at finding the ground state energy of spin lattices, these implementations are not particularly useful for studying holographic properties of tensor networks. For this purpose we created a method for simulating non homogeneous MERA networks to allow for full control over every individual tensor in the network. This project has successfully created a method of simulating entire MERA networks in order to find their ground state energy at a given transverse magnetic field strength. This is the first step in being able to study the holographic properties of these tensor networks as a toy model for discrete quantum gravity calculations.

Acknowledgements

I would like to thank Prof. Andrew Frey for the opportunity to conduct this research and for the supervision necessary to make this project possible.

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Bibliography

[1] G. Evenbly and G. Vidal. Algorithms for entanglement renormalization. *arXiv*, Jul 2007.