Universität Bonn

Compressing Neural Quantum States using Matrix Product Operators

Author:
Junaid Akhter
February 7, 2022

Supervisor:
Professor Matteo Rizzi

Contents

Chapter 1

Notations

Checked so far: 3.2 to 3.3.
Ready to be checked: 3.4 to 4.2
Color codes: I usually write the paragraphs in black but when there is supposed to be a change I do the following.

- Black: The text is good and needs minor changes (references etc)
- Red: The text is not completely correct and should be replaced
- Black: The text is supposed to replace the red text in the above point

Summary of a section/subsection

- The summary of the section is wtitten in bullet points under the respective section in green.

Can be ignored

- proper references
- spelling mistakes
- proper figures

However, comments are welcome

Chapter 2

Introduction

One of the main challanges that physicists are facing is solving the quantum many body problem. Quantum many body problem refers to the fact that the dimensionalilty of the *Hilbert Space* scales exponentially with respect to the system size. This means that as the size of the system increases, it becomes very difficult to diagonalize the Hamilton Matrix and hence find the eigenstates and the eigenvalues of the system.

Over the years physicists have developed many useful methods to extract the properties of many body systems some of which are Mean Field Theory, DMRG, DMFT, MC methods [references] etc. Almost all of these methods use some physical property of the system which saves us from studying the complete Hilbert Space and rather focus on some part it which are called physical states. For example with the help of tensor networks one is able to probe systems which follow the area law for their *entanglement structure*.

Recently Machine learning has gained a lot of attention in physics. Not only has ML found applications in computer vision, speech recognition, chemical synthesis [1] [references from review]but also in many scientific areas etc [references]. Also at the cross roads of quantum physics and machine learning a new exciting field call *quantum machine learning* as been born[references]. The field of QML can be classified into two broad categories, developing new quantum algorithms which can perform better than their classical counter part and the second one is where one uses classical machine learning tools to study quantum problems. The second interpretation of QML is going to be the focus in this work.

Carleo and Troyer in their paper [2] introduced a neural network ansatz for representing wave functions of quantum many body systems which are now called *neural quantum states*. They used a special kind of neural network called *Restricted Boltamann Machine (RBM)* which can be interpretted as a single layered feed forward neural network. To optimize the weights and biases (variatiional parameters) of the model they used *Stochastic Reconfiguration* rather than plain gradient descent. In this paper they demonstrated that neural quantum states can represent ground states effitiently and can be at par with the currently available numerical methods in the market e.g. Tensor Networks.

Since then a number of studies have been done along the same lines. One such study has been

done in studying the J_1, J_2 model [3]. A common problem that one encounters while optimizing parameters through *stochastic reconfiguration* is that it becomes computationally expensive if we have want to simulate larger systems die to increasing number of parameters.
We want to investigate the idea of using Matrix Product Operators for compressing [4] the RBMs as NQS. Given that RBMs are one of the simplest neural networks we want to investigate if we can compress the parameters or get a resonable accuracy while using fewer number of parameters in our model. Here, we show that such compression is possible and hence improving our ansatz from pure RBM to these hybrid architectures is favourable. We also show that using the hybrid architecture improves the performance of the pure gradient descent compared to the stochastic reconfiguration while using real variational parameters. Further we investigate which decomposition of the MPO leg dimensions favours convergence more easily.
Finally we present an outlook about how such architectures can be useful for both machine learning and tensor network community and what can one expect from them for future applications in quantum many body problems.

- Many Body Problem
- Methods to tackle it
- Using Machine learning in Many Body Physics
- Neural Quantum States
- Need for a compression of NQS
- Exploring the possibility of merging ideas from Tensor Networks and ML
- MPO used to compress NN
- Outlook

Chapter 3

Background

This chapter provides the necessary theoretical background for this work/report/thesis. We start by introducing the *quantum many body problem (QMBP)* and briefly mention some of the methods that physicists have devised over time to tackle it. We then introduce the *Neural Quantum States* (NQS) as an ansatz for representing the *quantum many body wave function (QMBWF)*, first introduced by Carleo and Troyer [2]. At the end of the chapter we discuss the challanges that one faces in optimizing the variational parameters if one wants to simulate larges systems, solving which is the motivation of this work/report/thesis.

3.1 Many Body Problem

Solving the QMBP is one of the most challenging problem that physicists have been trying facing since decades now. The problem precisely refers to the exponential scaling of the Hilbert Space in system size for a system of many quantum particles. From the principles of quantum mechanics, the wave-function of an N particle system where each particle has p possible states, can be described as follows:

$$|\Psi\rangle = \sum_{s_1,s_2\cdots,s_N=1}^{p} \psi\left(s_1, s_2 \cdots, s_N\right) |s_1\rangle \otimes |s_2\rangle \otimes \cdots \otimes |s_N\rangle \tag{3.1}$$

where the coefficients $\psi\left(s_1, s_2 \cdots, s_N\right)$ can be regarded as an N-variable complex function. As one can see, in order to describe a full QMBWF, we need to know the value of $\psi(\mathbf{s})$ corresponding to each possible configuration of $\mathbf{s} = (s_1, s_2 \cdots, s_N)$. Simple combinatorics reveals that there are p^N possible configurations of $\mathbf{s}$ i.e., the quantum many body wave-function of an N particle system requires $O(p^N)$ coefficients. With this dimensionality not only does it become impossible to solve these systems analytically but it is also very inefficient to solve such systems on a computer [1]. The direct consquence of such inefficiency is that we are not able to explore a lot of interesting phenomena that exist in nature since most of them appear in many body systems e.g., the mechanism behind high -T_c superconductivity, other importand condensed matter phenomena beyond Landau's

[1] So far the best that has been done is to simulate around 43 spijns at FZ Jülich[reference]

paradigm of phase transitions, topologically ordered phases, quantum spin liquids, deconfined quantum criticality etc are in principle many body phenomena.

To solve this problem physicists over the years have invented varried numerical methods to study quantum many body systems. One of the most powerful numerical methods is the quantum Monte Carlo (QMC) method [cite 1 in [3]]. The method is powerful in calculating the multi-dimensional integrals, which appear in many-body problems. Various physical quantities like total energy and correlation functions can be calculated exactly using QMC. One of the major drawbacks of QMC is that the method is not applicable when the negative-sign problem becomes severe in fermionic systems. The negative sign problem arises due to the antisymmeteric nature of the fermionic wave function.

Another class of methods which are very useful are the ones which are based on variational wave functions. In these methods the exact wave functions are approximated by some wave function ansatz determined by the so called variational parametes. One of the oldest variational method is the *Hartee Fock* (HF) method which is used to approximate the state and the energy of a stationary state [needs reference]. Most of these methods use the property that physical states belong to a much smaller space of the total Hilbert Space e.g., density matrix renormalization group (DMRG) [6,7 in [3]] and tensor network methods [8, 9 in [3]] can approximate states which follow area law [reference needed] over entanglement which means that the entanglement between two subsystems is proportional to the boundary between the sysems rather than the volume. A widely known limitation of tensor network approaches is that they it becomes very difficult to efficiently contract them in two dimensions.

Recently Carleo and Troyer [2] have proposed a variational wave function based to represent QMBWF based on Deep Neural Networks (DNN). They used a special neural network called restricted Boltzmann machine (RBM) [11 in[3]] to represent the QMBWF. The weights and biases of the RBM serve as the variational parameters of the model. In this way the RBM can be considered as particular form of the variational wave function in the *variational Monte Carlo* (VMC) method. One key advantage of using NQS like RBM over tensor network is that it can represent states with volume law entanglement entropy [13, 14 in [3]]. In the following section we will introduce the NQS ansatz in detail.

Some of these approaches are stochastic for example QMC which rely on probabilistic approaches which demands positive wave-function [1-3 in Carleo and Troyer]. Compared to this there are compression approaches, where one tries to write down an efficient representation of the wave-function for example in terms of Tensor Networks[7,8 from Carleo], MPS [4-6 from Carleyo] being the most common one for 1-D gapped hamiltonians. There are however drawbacks with both kinds of approaches. While the former suffers from the sign problem [9 in troyer], the latter is difficult to contract in high-dimensional systems. This leaves a wide variety of regimes unexplored which also include many interesting open problems. In other words the many body problem boils down to the inability of finding a general strategy to extract the useful information/features from the

exponentially large Hilbert Space.

- Exponential growth of many body systems
- The most interesting phenomena are mostly Many Body Phenomena
- Some methods that have been devised to study Many Body Systems
- Neural Quantum States as wave-function ansatz

UPDATE AFTER FIRST DRAFT

3.2 Neural Quantum States (NQS)

After their introduction NQS have found applications in studying a variety of problems. They have been used to study both quantum spin systems without geometrical frustation [2] and with geometrical frustation [15-21 in [3]], fermion systems [15, 24-29 in [3]] topological states [13, 31-36 in [3]], non-equillibrium states [5], excited states [21, 30, 37, 38], quantum systems with nonabelian or anyonic symmeteris [38 in [3]] etc. Since NQS are based on the concepts from *deep learning*, we are going to dive in a gentle introduction to deep learning.

3.2.1 Deep Learning (DL)

Deep Learning (DL) is one of the most exciting and dynamic areas of modern research and application. In order to keep it simple, we will discuss the most basic form of learning known as *supervised learning*.
Imagine a setting where we are given a dataset $D = (\mathbf{x_1}, y_2), (\mathbf{x_2}, y_2), ..., (\mathbf{x_N}, \mathbf{y_N})$ where $\mathbf{x_n} \in \mathrm{R}^D$ is a vector and $y_n \in \mathrm{R}$ are the corresponding labels. An example would be $\mathbf{x}_n$ being the pixels of images of dogs and cats and y_n being a number which tells us whether the image is of a dog or a cat (say $0 \rightarrow$ dog and $1 \rightarrow$ cat). We wish to estimate a predictor $f(\cdot, \boldsymbol{\theta}) : \mathrm{R}^D \rightarrow \mathrm{R}$ where $\boldsymbol{\theta}$ are the parameters of the predictor. The word *learning* in DL referes to finding the right set of parameters $\boldsymbol{\theta}^*$ such that we fit the data well, i.e.,

$$f(\mathbf{x_n}, \boldsymbol{\theta^*}) \approx y_n \qquad \text{for all } n = 1, ..., N. \tag{3.2}$$

In this way our predictor has succesfully learned the features of input data $\mathbf{x}$ (cats and dogs). After learning is done, it should be able to efficiently categorize unseen (images not used during learning) images of dogs and cats. For that purpose It is a usual practice to split the dataset into a training dataset D_{train} and a test dataset D_{test}. As is clear by their names, the network is trained on D_{train} and then its performance is tested on D_{test}. Following are the key components of performing a such a task using DL. In general following three ingredients are necessary to perform such a task

- *A Predictor/Model:* A model is needed to make a prediction or classification of the input data. The data is labelled in *supervised learning* but that is not necessary the case e.g., in *unsupervised learning*.
- *Loss function*: To account for how well our model is making the prediction $\hat{y_n} = f(\mathbf{x_n}, \boldsymbol{\theta^*})$ based on the input $\mathbf{x_n}$, we need to specify a loss function $l(y_n, \hat{y_n})$. This loss function takes the ground truth y_n and the prediction $\hat{y_n}$ and returns a non-negative number representing how much error the model has made on this particular prediction. In the example given above, it will tell us how many times our model has assigned a picture to a wrong label i.e., identifies a dog as a cat or vice versa.
- *An Optimization Process*: Given that we have defined a loss function l to give us a measure of how well our model is fitting the data, we want to iteratively optimize our parameters $\boldsymbol{\theta}$ in such a way so that we minimize l.

In DL the predictor/model $f(\mathbf{x_n}, \boldsymbol{\theta^*})$ is a *Deep Neural Network* (DNN). A DNN is constructed using *Artificial Neural Networks* (ANN) whose basic building block is a neuron which takes a d-dimensional vector $\mathbf{x}$ as an input and spits out a scalar $a_i(\mathbf{x})$. Generally one can decompose a_i into a linear operation followed by a non-linear transformation $\sigma_i(z)$ (see figure 3.1).

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)}) = \sigma_i(\boldsymbol{w}^{(i)}.\mathbf{x} + b^{(i)}) \tag{3.3}$$

where $\boldsymbol{w}^{(i)} = \left(w_1^{(i)}, w_2^{(i)}, ..., w_d^{(i)}\right)$ are the weights corresponding to the i_{th} neuron and $b^{(i)}$ is the neuron-specific bias. Together weights and biases, $\mathbf{w}^{(i)} = (b^{(i)}, \boldsymbol{w}^{(i)})$ are known as the model parameters. The non-linearities σ_i in a model can be chossen from a variety of choices like sigmoids (i.e., Fermi functions), the hyperbolic tangent etc. A DNN is constructed by stacking different layers or neurons one after the another with the continuity that the output of one layer serves as an input to the next layer (see figure 3.1 right).

Once we have a model which produces an output corresponding to a certain input, the next thing to do is to use a well defined optimization method to minimize the cost function, $l(y_n, \hat{y_n})$. This is usually done using the *gradient descent* (GD) [reference] algorithm where the variational parameters of the model are updated as

$$\theta \rightarrow \theta' = \theta - \tau \frac{\partial l}{\partial \theta} \tag{3.4}$$

where τ is a hyperparameters of the model known as the learning rate.
Eqn.(3.4) means that the variational parameters are updated in a manner so that at each iteration we are moving towards the minimum of the cost function l. Intuitively one can think of the cost

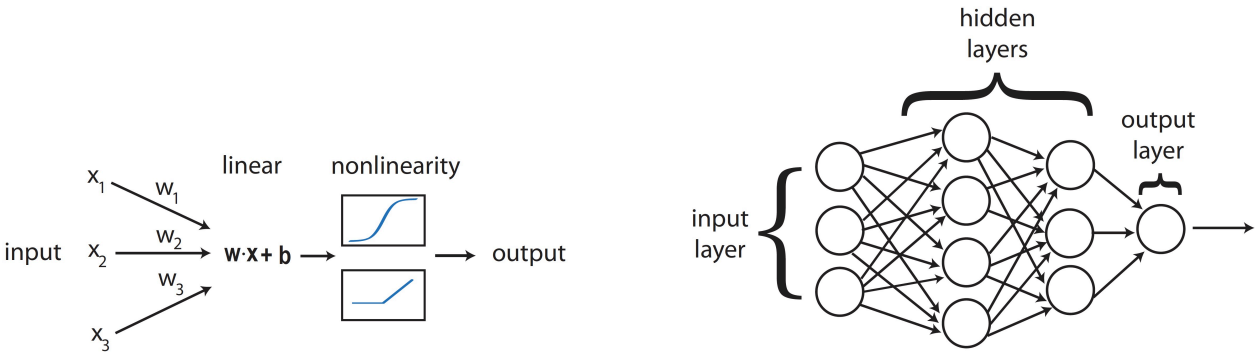


Figure 3.1: (left) The basis components of a neural network which consists of a linear transformation and followed by a non-linear transformation; (right) Neurons are stacked into layers with the output of one layer serving as the input to the next layer.[Reference from Marin]

function as a landscape (figure 3.2) where a particular point is marked by the model parameters. The goal of the optimization/learning is to go down this energy landscape until we reach the lowest point.

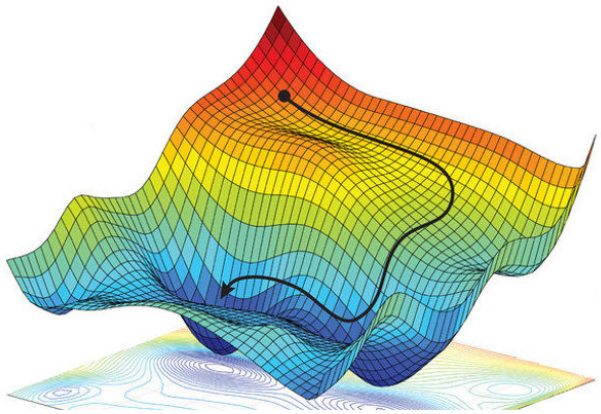


Figure 3.2: source of image

The magic behind the working of neural networks lies in the *universal approximation theorem* which states that a NN with a single layer of neurons can approximate any continuous, multi-input/multi-output function with arbitrary accuracy[reference from Markus & Markus 35-37]. However, it has been observed that upon increasing the depth of the neural network the representational power of the neural network is greatly expanded. The freedom of choosing the number of layers, type of non-linearity etc. makes it possible to create a variety of architectures e.g., *concolutional neural networks* (CNNs), *Autoencoders*, *restricted Boltzmann machines* (RBM) etc.

3.2.2 Representation of Many Body Wave function as Artificial Neural Neural Networks

Over the yers DNN have made their way into a variety of fields and physics is no acception. Here we want to see how DNN can be used to represent a quantum many body wave function. In this

work we are going to work with a special kind of NN, restricted Boltzmann machine (RBM) as was also done in the original work [2].

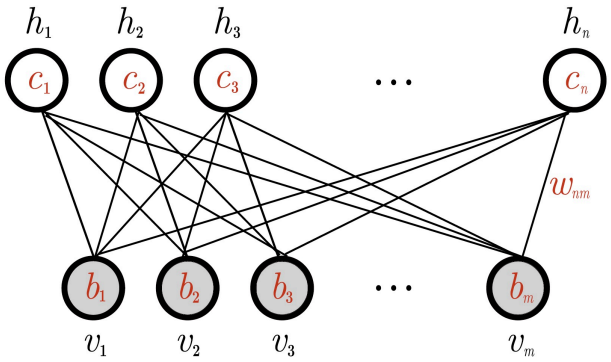


Figure 3.3: Interaction between hidden and visible units [notes]

RBM belongs to a special class of NN known as energy based models (for a detailed account on RBM check [notes]). A RBM is a parametric model of a joint probability distribution between two variables, $\mathbf{h} = h_1, h_2, \cdots h_m$ and $\mathbf{v} = v_1, v_2, \cdots, v_n$ with the probability given by

$$p(\mathbf{h}, \mathbf{v}) = e^{-E(\mathbf{h},\mathbf{v})} \tag{3.5}$$

where

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{M}\sum_{j=1}^{N} W_{ij} h_i v_j - \sum_{i=1}^{M} b_i h_i - \sum_{j=1}^{N} a_j v_j \tag{3.6}$$

and $\mathbf{v}$ and $\mathbf{h}$ are known as visible and hidden units respectively. Each visible unit v_i is connected to each hidden unit h_j through a coupling W_{ij} as defined in Eqn. 3.6. However, there is no intra-layer interaction within the visible or hidden layers (see figure 3.3). Together W_{ij}, b_j and c_i are the model parameters and are usually denoted by $\theta = \{W_{ij}.b_j, a_i\}$. By identifying the visible units with the observable (e.g., spin states $v_i \to s_i$) and $\boldsymbol{\theta}$ with the variational parameters Eqn. 3.5 is re-defined as the varational wave-function.

$$\psi_{\boldsymbol{\theta}}(\mathbf{s}) = \sum_{\{h_i\}} e^{\sum_{ij} W_{ij} h_i s_j + \sum_i b_i h_i + \sum_j a_j s_j} \tag{3.7}$$

where we have identified $v_i \to s_i$, spin state of the i_{th} spin. As there are no intra-layer interactions. we can trace out the hidden layers. By restricting $h_i = \{-1, 1\}$, we can write Eqn. 3.7 is a more compact form.

$$\psi_{\boldsymbol{\theta}}(\mathbf{s}) = e^{\sum_i a_i s} \times \prod_{i=1}^{M} 2\cosh[b_i + \sum_j W_{ij} s_j] \tag{3.8}$$

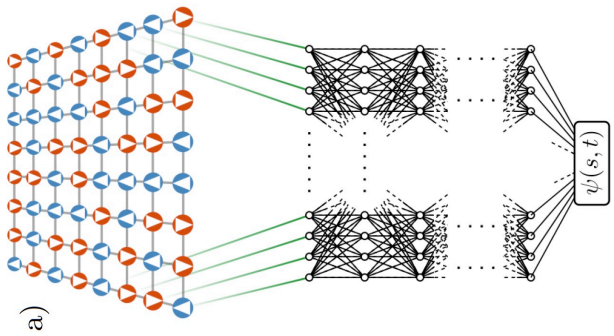


Figure 3.4: NQS representation of a many body wave-function

What Eqn. 3.7 tells us is that for each input spin configuration $\mathbf{s}$ and pre-defined θ, our network maps it to a complex number $\psi(\mathbf{s}, \boldsymbol{\theta})$ (see figure 3.4). Given that we have access to $\psi(\mathbf{s}, \boldsymbol{\theta})$, we can write the variational wave-function of a many body system in the follwoing way.

$$|\psi(\theta)\rangle = \sum_{\mathbf{s}\in\mathbb{Z}_p^N} \psi(\mathbf{s}, \theta)|\mathbf{s}\rangle \tag{3.9}$$

Eqn. 3.9 is used as an ansatz for the variational wave-function in VMC.

3.2.3 Varational quantum montecarlo

Varational Monte Carlo (VMC) [reference needed] method is a quantum MC method used to find ground states of quantum systems. The methods requires a trial wave-function also known as variational ansatz which is dependent on some adjustable parameters known as variational parameters. An optimization scheme is used to update the variational parameters so that at the end of optimization a very good approximation of the ground state is obtained.
Given that we have acces to a variational wave-function from Eqn. 3.8, the expectation value of a local operator $\hat{O}$ can be calculated as follows:

$$\frac{\langle\psi(\theta)|\hat{O}|\psi(\theta)\rangle}{\langle\psi(\theta)\mid\psi(\theta)\rangle} = \sum_{\mathbf{s},\mathbf{s}'} \frac{\psi_\theta(\mathbf{s})^*\psi_\theta(\mathbf{s}')}{\langle\psi(\theta)\mid\psi(\theta)\rangle}\left\langle\mathbf{s}|\hat{O}|\mathbf{s}'\right\rangle \tag{3.10}$$

$$= \sum_{\mathbf{s}} \frac{|\psi_\theta(s)|^2}{\langle\psi(\theta)\mid\psi(\theta)\rangle}\sum_{\mathbf{s}'}\left\langle\mathbf{s}|\hat{O}|\mathbf{s}'\right\rangle\frac{\psi_\theta(\mathbf{s}')}{\psi_\theta(\mathbf{s})} \tag{3.11}$$

$$\tag{3.12}$$

The term $|\psi_\theta(\mathbf{s})|^2/\langle\psi(\theta)\mid\psi(\theta)\rangle$ can be readily interpretted as a probability distribution $p_\theta(\mathbf{s})$ on the computaional basis.

$$\frac{\langle\psi(\theta)|\hat{O}|\psi(\theta)\rangle}{\langle\psi(\theta)\mid\psi(\theta)\rangle} = \sum_{\mathbf{s}} p_\theta(\mathbf{s})O^\theta_{\text{loc}}(\mathbf{s}) \equiv \left\langle\left\langle O^\theta_{\text{loc}}\right\rangle\right\rangle_\theta \tag{3.13}$$

where we introduced the local estimator

$$O^{\theta}_{\mathrm{loc}}(\mathrm{s}) = \sum_{\mathrm{s}'} \left\langle \mathrm{s}|\hat{O}|\mathrm{s}' \right\rangle \frac{\psi_\theta(\mathrm{s}')}{\psi_\theta(\mathrm{s})} \tag{3.14}$$

and the notation $\langle\langle . \rangle\rangle$ stands for an expectation value with respect to $p_\theta(\mathrm{s})$. As we can see there are two sums in Eqn. 3.12 and both of them could become infeasible over large systems N. However, typical operators are sparse in the computational basis which means that the sum $\sum_{\mathrm{s}'}$ can be evaluated efficiently. By contrast, the sum over all basis states, $\sum_{\mathrm{s}'}$ becomes infeasible because of the exponential growth of the Hilbert Space. At this point one has to resort to Monte Carlo (MC) sampling of $p_\theta(\mathrm{s})$ to efficiently estimate these expectation values. Using *Metropolis algorithm* (Explain in appendix maybe) we can obtain samples $\{ \mathbf{s}^{(1)}, \mathbf{s}^{(1)}, ...\mathbf{s}^{(N_{\mathrm{MC}})} \}_{\mathbf{s}\sim p_\theta((s))}$ and the expectation values can be approximated by the emperical mean as follows

$$\left\langle \left\langle O^{\theta}_{\mathrm{loc}} \right\rangle \right\rangle_\theta \approx \frac{1}{N_{\mathrm{MC}}} \sum_{n=1}^{N_{\mathrm{MC}}} O^{\theta}_{\mathrm{loc}} \left(\mathbf{s}^{(n)} \right) \tag{3.15}$$

- Definition of Variational monte carlo
- Metropolis Algorithm
- Calculating the expectation value of an operator using VMC
- for this purpose, only the functional form of $\psi_t heta(\mathrm{s})$ must allow for efficient evaluatioin [reference]
- given that we need to do sampling RBM can be sampled easily.

Ground state search

Ground state of a system corresponds to a state which has the minimum energy. Using VMC we can obtain the ground state by minimising the energy expectation value given by

$$E(\theta) = \frac{\langle \Psi(\theta)|\hat{\mathrm{H}}|\Psi(\theta)\rangle}{\langle \Psi(\theta)|\Psi(\theta)\rangle} \tag{3.16}$$

In DL language $E(\theta)$ is our cost function l and we have to use an optimization method to minimize it. However, we do not have access to labels as was the case when we discussed *supervised learning*. Hence, this form of learning the ground state of a system is not a *supervised learning* task. Neverthless, to search the minima in the energy landscape GD algorithm could be used. We have to calculate the gradients of $E(\theta)$ with respect to the variational parameters θ_k at each iteration of the optimization step k. This can be easily calculated from Eqn. 3.16

$$\nabla_{\theta_k} E(\theta) = 2\mathrm{Re}\Big(\langle\langle (O^{\theta}_k)^* E^{\theta}_{loc} \rangle\rangle_\theta - \langle\langle (O^{\theta}_k)^* \rangle\rangle_\theta \langle\langle E^{\theta}_{loc} \rangle\rangle_\theta \Big) \tag{3.17}$$

where $E^{\theta}_{loc}(\mathbf{s}) = \sum_s \langle \mathbf{s}|\hat{H}|\mathbf{s}'\rangle \frac{\psi_\theta(\mathbf{s}')}{\psi_\theta(\mathbf{s})}$ is defined as in Eqn. 3.14 is the local Energy. Further we also introduced the logarithmic derivatives $O^{\theta}_k(\mathbf{s}) = \partial_{\theta_k} \log \psi_\theta(\mathbf{s})$ which basically appear due to the multiplication of a unity $\psi_\theta(\mathbf{s})/\psi_\theta(\mathbf{s})$ in order to obtain the factors of $|\psi_\theta(\mathbf{s})|^2$. A gradient

based optimization would mean that we update the variational parameters in the following iterative fashion.

$$\theta_k^{(n+1)} = \theta_k^n - \tau \partial_{\theta_k} E(\theta)|_{\theta=\theta^n} \tag{3.18}$$

However, itt has been observed that the energy landscape $E(\theta)$ is not convex i.e., the landscape has local minimas and saddle points. A very common drawback of using plain GD is that during optimization it is prone to getting stuck in these unwanted territories which lead to very slow convergence. To avoid this modified versions of GD are used for optimization. *Stochastic Reconfiguration* (SR) method [reference] is one such scheme, where the update is modified to

$$\theta_k^{(n+1)} = \theta_k^n - \tau \mathcal{S}^{-1}_{k,k'} \partial_{\theta_{k'}} E(\theta)|_{\theta=\theta^n} \tag{3.19}$$

where $S_{k,k'} = \langle\langle (O_k^\theta)^* O_{k'}^\theta \rangle\rangle_\theta^c$ is called the *quantum Fisher matrix*, which is the metric tensor of the Fubini-Study metric- a natural metric on the projective Hilbert space [reference]. Another way of putting SR is that SR methods updates the variational parameters according to the GD weighed by the quantum Fisher matrix. It provides informatioin about the local geometry of the local variational manifold, which is exploited by adjusting the gradient step accordingly in Eq. (3.19).

3.3 The curse of dimensionality

A very common problem which plagues the ML community is the problem of *curse of dimensionality*. This pertains to the difficulty in optimising the increasing number of parameters in deep neural networks as the networks become larger [references]. Also a neural network with large number of parameters is prone to overfitting [4]. NQS are no exception to it and as one increases the number of hidden units in RBM, the optimization becomes very costly. This is particularly the case when one wants to simulate systems with more degrees of freedom (lerger systems or 2-D systems). Although N_{var} scales polynomially in the system size [reference], it becomes very difficult to optimize for larger systems [reference]. This problem becomes more prominent when one uses *stochastic reconfiguration* as one has to invert the S matrix in equation 3.19 which in an expensive operation. All in all this is a stumbling block which restricts one to investigate the efficiency of the RBM wave functions in larger systems. One such attempt has been made where RBM and the Gutzwiller-projected fermion wave function is combined[18, 20 in] [3]]. In this thesis we would like to approach this problem from a different perspective as discussed in the next chapter.

3.4 Parameter scalling in NQS

We saw in Eqn (3.1) that the dimensionality of the Hilbert Space blows up exponentially with the system size. Using Eqn. (3.8) the number of parameters in NQS is $N_{\text{var}} = \alpha N^2$ ($\alpha = M/N$). By increasing the number of hidden units M, we increase the number of the variational parameters

and hence the accuracy of the model [reference figur from carleo]. As can be seen the number of variational parameters in NQS scale polynomially ($\sim N^2$) with the system size. This is the main goal of our work.

$$N_{\text{var}} \sim N^2 \tag{3.20}$$

This is already an efficient representation of the full wave function. However, even with this scaling the optimization becomes difficult when one wants to investigate large system sizes [3]. This problem becomes more prominent with the SR optimization scheme where one has to invert the S matrix in Eqn. (3.19) which in an expensive operation. Therefore it becomes crucial to reduce N_{var} if we want to study larger system sizes. We want to write a modified RBM ansatz which is compressible in N_{var}.

- $$N_{var} \sim N_{site}^2$$
- Expectation value of Energy
- Optimization using Simple Gadient descent
- Stochastic Reconfiguration
  - Quantum Fisher Matrix

Comments on this chapter

- Write more about SR method maybe.

Chapter 4

Compressing variational parameters inside NQS

As pointed out at the end of the previous chapter, it is crucial to reduce N_{var} in order to be able to simulate larger system sizes. [18, 21 in [3]] discuss one way where the RBM wave function has been combined with the Gutzwiller-projected fermion wave functions. In this work we would like to address the same problem in a different way.
We write a modified RBM wave-function with the modification that we replace the full weight matrix W (which contains the variational parameters) with a *Matrix Product Operator*. This idea was used in [4] for supervised learning with image data and it was shown that one could do a compression in N_{var} without compromising on the performance of the model. However, such modified models have never been explored in the context of NQS. We would like to explore this possibily of compressing N_{var}. In the following we want to give a gentle introduction to MPO followed by how we can create hybrid architectures from them together with RBM.

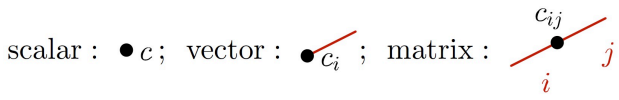
- Need to reduce N_{var}
- Mention recent works in the direction of combined architectures from tensor networks and Neural Networks.
- Mentions that one such idea which has not been explored in the context of NQS is the idea of using MPO iniside NN.
- Such architectures have not been used in NQS

4.1 A brief intoduction to Matrix Product Operators

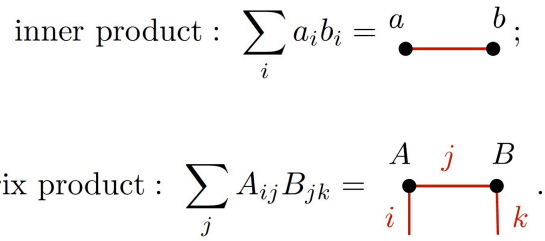
MPO are a class of *Tensor Networks* (TN)[1] which were first introduced in physics to characterize the short-range entanglement in one-dimensional quantum many body systems [34-35 in [4]]. A TN is a factorization of a higher-order tensor into a sequential product of the so-called local tensors.
A rank-n tensor is defined as a complex variable with n indices, for example $T_{i_1,i_2,\cdots i_n}$. The number

[1] also known by the name tensor-train decomposition in mathematic literature

of values that a pariticular index i_k can take is the dimension of index i_k. In this way the simplest tensor that we can write is a scalar i.e., a tensor of rank zero. Similarly a vector V_k is a tensor of rank one and a matrix, $M_{i,j}$ is a tensor of rank two. It is very convenient to visualize tensors graphically known as *tensor network diagram*. A rank-n tensor is represented as a vertex with n edges. Thus a scalar is just a vertex, a vector is a vertex with on edge and a matrix is a vertex with two edges as shown



The graphical interpretation comes handy especially if we want to do a contraction between indices. This is represented by connecting two vertices which have the same edje label. For example an inner product between two vectors and a matrix product between a matrix and a vector are represented as



In a similar fashion one can imagine that it is very convenient to represent complicated contractions over many indices with TN graphs.

A MPO is one of the many possible tensor network where a complicated tensor like $M^{i_1,\cdots i_n}_{j_1,\cdots,j_n}$ can be written as a contraction over smaller tensors , $\Lambda^{[1]}\cdots\Lambda^{[n]}$

$$M^{i_1,\cdots i_n}_{j_1,\cdots,j_n} = \sum_{\{\chi\}} \Lambda^{[1]i_1\chi_1}_{j_1} \Lambda^{[2]i_2\chi_2}_{\chi_1 j_2} \cdots \Lambda^{[n]i_n\chi_n}_{j_n} \tag{4.1}$$

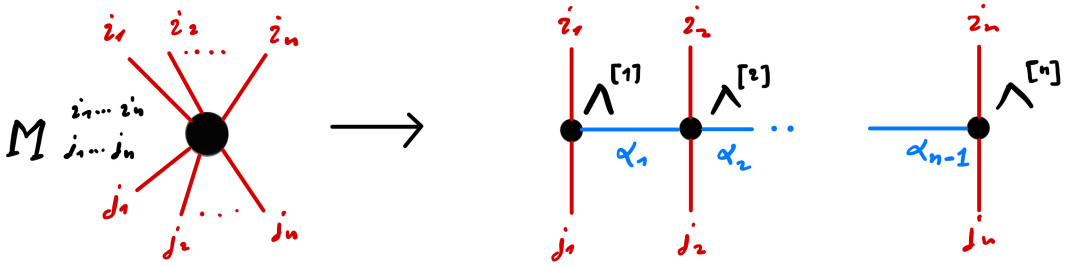


Figure 4.1: Decomposing a multidimensional tensor into an MPO. Reminder: blue lines have to be called α

where $\Lambda^{[k]i_k}_{j_k}$ is a $\alpha_{k-1}\times\alpha_k$ matrix with α_k as the dimension of the bond connecting $\Lambda^{[k]}$ and $\Lambda^{[k+1]}$ (usually known as the bond dimension). The first and the last tensors in the MPO are three legged ($\chi_0 = \chi_1 = 1$) and the rest in between are four legged tensors. One interesting property of this

decomposition is the scaling of N_{var}. If I_k and J_k are the dimensions of i_k and j_k then the number of parameters in $M^{i_1,\cdots i_n}_{j_1,\cdots,j_n}$ are

$$N^{\text{M}}_{\text{var}} = \prod_{k=1}^{n} I_k \times \prod_{l=1}^{n} J_l \tag{4.2}$$

Compared to this the total number of variational parameters in MPO equals

$$N^{\text{mpo}}_{\text{var}} = \sum_{k=2}^{n-1} I_k J_k \chi^2 + I_1 J_1 \chi + I_n J_n \chi \tag{4.3}$$

which could be a great reduction compared to $N^{\text{M}}_{\text{var}}$ if χ is can be choosen sufficiently small.

- Introduction to Tensor Networks and MPO
- Comparison of the number of variational parameters of MPO and a multidimensional tensor
- Should we write how we can get an MPo from the multidimensional tensor from successive SVD?

4.2 Modified NQS ansatz

To write a compressed NQS we write a representation where the full weight matrix is replaced by a MPO. This can be achieved as follows: First we need to do the follwoing reshaping

$$W_{sh} \longrightarrow W^{i_1 i_2 \cdots i_n}_{j_1 j_2 \cdots j_n} \tag{4.4}$$

with the follwoing constraint

$$\prod_{k=1}^{n} I_k = M, \qquad \prod_{l=1}^{n} J_l = N \tag{4.5}$$

where I_k, J_l are defined as before and M and N are the number of visible and hidden units. The one-dimensional coordinate s of the input configuration/vector $\mathbf{s}$ with dimension N is reshaped into a coordinate in an n-dimensional space, labeled $(j_1 j_2 \cdots j_n)$. Similarly the $\mathbf{h}$ coordinate with dimension M , is reshaped into $(i_1 i_2 \cdots i_n)$. Hence, there is a one to one mapping between s, h to their n-dimensional spaces, $(j_1 j_2 \cdots j_n)$ and $(i_1 i_2 \cdots i_n)$ respectively.
Finally $W^{i_1 i_2 \cdots i_n}_{j_1 j_2 \cdots j_n}$ can be represented as a MPO similar to Eqn. (4.1) and as shown is figure (4.1). Pictorially the whole process can be shown as in figure (4.2). Using this decomposition of the W matrix, the modified RBM wave-function can be written as

$$\psi_\theta(\mathbf{s}) = e^{\sum_{i=1}^{N} a_i s_i} \prod_{i_1=1}^{I_1} \prod_{i_2=1}^{I_2} .... \prod_{i_n=1}^{M/I_1 I_2 ... I_{n-1}} 2\cosh\left(\chi_{i_1 i_2 .... i_n}\right) \tag{4.6}$$

where

$$\chi_{i_1 i_2 .... i_n} = \sum_{\{\alpha\}} \sum_{j_1, j_2, .. j_n} \Lambda^{[1],i_1}_{j_1,\alpha_1} \Lambda^{[2],i_2}_{j_2,\alpha_1\alpha_2} ... \Lambda^{[n],i_n}_{j_n,\alpha_{n-1}} \sigma_{j_1 j_2 ... j_n} \tag{4.7}$$

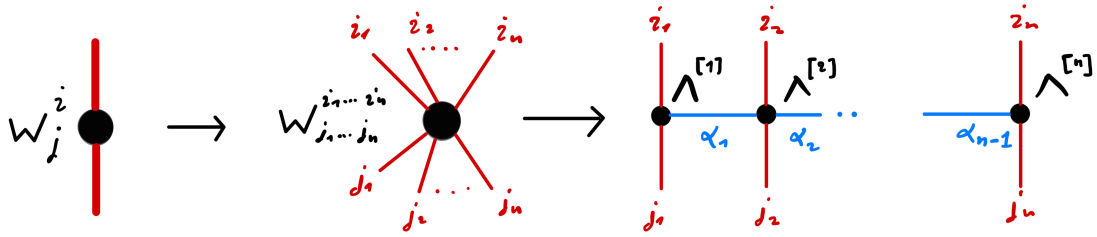


Figure 4.2: Decomposing a multidimensional tensor into an MPO.

where the symbols have their usual meaning and the input spin vector **s** has been reshaped into $\sigma_{j_1 j_2 \ldots j_n}$ as shown in figure (4.3). This is important in order to make the contraction over indices $(j_1, j_2..j_n)$ possible.

It is important to notice that the index decomposition in Eqn.(4.4) is neither possible for all system sizes nor unique. Only a system size which can be factorized into smaller numbers can be reshaped and hence decomposed into an MPO. For example there is no way to decompose a system size of prime numbers into an MPO. Further a system size which is decomposible can be decomposed in many possible ways. Similarly we can choose the number of hidden units, M in such a manner that it can be decomposed in multiple ways. For example for a system size of 16 (N = 8) and $\alpha = 2$ (N = 16) we can reshape W into $W^{4,2}_{8,2}$, $W^{2,4}_{8,2}$, $W^{4,2}_{4,4}$, $W^{2,4}_{4,4}$ and $W^{2,2,2}_{2,2,2}$.

Now using 4.6 as our wave-function ansatz we use variational MC along with the SR to estimate the ground state of TFIM with OBC. The results are discussed in chapter 5.

- Pictorial and mathematical representaion of the new ansatz
  - Constrainst about which system sizes can be decomposed into an MPO
- Important to point out that we can do the MPO decomposition even after obtaining W but that will not help in optimization since we are already using the full W
- How this strategy is different from the previous one
- How gradients over this ansatz look.
- Entanglement entropy of the input data with respect to the bond dimension

4.2.1 Proper initialization of the network

One of the very common problems that plagues the gradient based optimization of DNN is the *vanishing gradient problem* or *barren plateau problem*. In a DNN gradients are found by using the *back propagation algorithm* which basically uses the chain rule by multiplying the derrivatives from the outer layer to the inner layer to compute the derrivatives of the initial layer. This causes the gradient to decreases exponentially with increasing depth of the neural network. The problem of vanishing

gradients leads to convergence of the cost before it has reached the minimum value[reference to apendix].

In a RBM there is only a single layer of neurons and hence we do not face the vanishing gradient problem. However, we encountered this problem with our modified ansatz. The vanishing gradients of energy, $\nabla_{\theta_k} E(\theta)$ in Eqn.(3.19) are a consquence of vanishing derrivates of wave-function, $O_k^\theta(\mathbf{s}) = \partial_{\theta_k} \log \psi_\theta(\mathbf{s})$. To understand this more clearly let us write down the equation for the gradient of the logarithm of the wave-function.

$$\frac{\partial \log \psi_\theta(\mathbf{s})}{\partial \Lambda^q_{kp}} = \sum_{i_1=1, i_2=1 \ldots, i_n} \left(\frac{\sinh\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\cosh\left(\chi_{i_1,i_2 \ldots i_n}\right)} \right) \frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda^q_{kp}} \tag{4.8}$$

$$= \sum_{i_1=1, i_2=1 \ldots, i_n} \tan\left(\chi_{i_1,i_2 \ldots i_n}\right) \frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda^q_{kp}} \tag{4.9}$$

where $\chi_{i_1,i_2 \ldots i_n}$, as in Eqn. (4.7) is obtained after contraction over all the physical indices $j_1, j_2, \cdots j_n$ and the bond dimensions $\alpha_1, \alpha_2 \cdots \alpha_{n-1}$ (figure 4.3 left). The derrivative of $\chi_{i_1,i_2 \ldots i_n}$ with respect to the entries of the first node of the MPO i.e., $\frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda^q_{kp}}$ is shown on the right. The multiplication of numbers during contractions from different nodes leads to the vanishing gradient problem in our case. Due to this our network is not able to learn the ground state and would get stuck in some local minima a few iterations after the initialization.

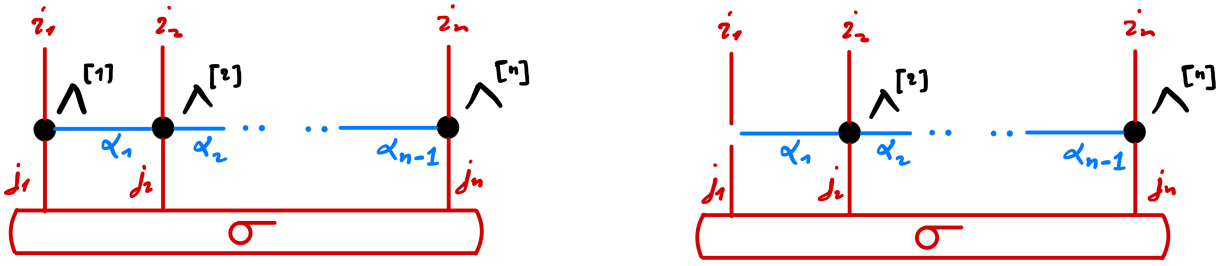


Figure 4.3: (left) Contraction of the MPO with the reshaped input configuration $\tilde{\sigma}$ i.e., $\chi_{i_1,i_2 \ldots i_n}$; (right) derrivative of $\chi_{i_1,i_2 \ldots i_n}$ w.r.t $\Lambda^{[1]}_{i_1,j_1,\alpha_1}$.

To avoid this problem we have to initialize our network more carefully rather than any random distribution. One solution would be to use the Xavier initialization as discussed in [6]. This however, only works for the case when we have real parameters. A more carerful analysis is needed while initializing with the complex parameters which is the case that we are interested in since the wave-functions in general are complex valued.

For a proper initialization of the network we look at the variances of the gradients of the wave-functions i.e., $\mathrm{Var}\left[O_k^\theta(\mathbf{s})\right]$. By initializing our network parameters in such a manner so that $\mathrm{Var}\left[O_k^\theta(\mathbf{s})\right]$ is non-zero and finite, we can make sure that our gradients $O_k^\theta(\mathbf{s})$ do not vanish. In the following we derrive an approximate expression for $\mathrm{Var}\left[O_k^\theta(\mathbf{s})\right]$ in terms of the variances of the individual nodes of the MPO.

Now, to calculate Var $\left[O_k^\theta(\mathbf{s})\right]$ from Eqn. (4.9) which includes a sum over indices $i_1, i_2, \cdots, i_n$, we can use the following property

$$\begin{aligned} \text{Var}\left(\sum_{i=1}^{n} X_i\right) &= \sum_{i=1}^{n} \text{Var}\left(X_i\right) + 2\sum_{i<j} \text{Cov}\left(X_i, X_j\right) && (4.10)\\ &= \sum_{i=1}^{n} \text{Var}\left(X_i\right) && (4.11)\\ &= n\,\text{Var}\left(X\right) && (4.12) \end{aligned}$$

Since we are initializing our networks in a random fashion, we have ignored the second term in the first step. In the final step we have used the law of large random numbers which says that the sum of the variance of n-random variables is equal to n times the variance of a single random variable. Further since the contraction of MPO involves multiplication of the random number, we can use the following property.

$$\begin{aligned} \text{var}\left(X_1 \cdots X_n\right) &= \prod_{i=1}^{n}\left(\text{var}\left(X_i\right) + \left(E\left[X_i\right]\right)^2\right) - \prod_{i=1}^{n}\left(E\left[X_i\right]\right)^2 && (4.13)\\ &= \prod_{i=1}^{n}\left(\,\text{var}\left(X_i\right)\right) && (4.14) \end{aligned}$$

where we have ignored $E[X_i]$ since we initialize our network with zero mean. Finally, to simplify the calculation further, we approximate $\tan\chi \sim \chi$.

Using the above properties we get an equation for the Var $\left[O_k^\theta(\mathbf{s})\right]$ in Eqn.(4.9) for each node in terms of the variances of nodes and the dimensions of the edges of the MPO. By fixing Var $\left[O_k^\theta(\mathbf{s})\right]$ to a desired number (say v) we can solve these n equations to derrive the individual variances of the MPO nodes. A detailed calculation is shown in the appendix [reference].

$$V_k = \begin{cases} \frac{1}{J_k}\left(\frac{\prod_{l=1}^{n} I_l v}{I_k^{2n-1}\alpha}\right)^{1/(2n-1)} & ;\text{ If } k=1 \text{ or } k=n \\ \frac{1}{J_k}\left(\frac{\prod_{l=1}^{n} I_l v}{I_k^{2n-1}\alpha^{2n}}\right)^{1/2n-1)} & ;\text{ otherwise} \end{cases} \qquad (4.15)$$

where V_k is the variance with which we should initialize the k_{th} node of MPO, J_k and I_l have the same meaning as before. α is the bond dimension of the MPO which we have assumed to be the same throughout.

Now that we know the variance with which each node should be initialised, we can start our network such that it does not face the problem of the vanishing gradient. However, in a computer we have to initialize the real and the complex parts of the seperately and use the follwing property to get the variance of the complex random number.

$$\text{var}\left[Z\right] = \text{var}\left[\text{Re}\left(Z\right)\right] + \text{var}\left[\text{Im}\left(Z\right)\right] \qquad (4.16)$$

Eqn.(4.15) is a very useful result when it comes to creating hybrid architectures from TN and DNN. Following the same procedure one can derrive an initialization condition for any other architecture where MPO is incorporated.

- Why does we face this problem for complex parameters in general
- The gradients of the cost with respect to the parameters are too small, leading to convergence of the cost before it has reached the minimum value
- Vanishing gradient/barren plateau problem
- Where does it arise from in our architecture
- Solution is to set a non-zero variance
- Tensor derrivative with a picture
- Assumptions and steps to a proper initialization
- An assumption that I have not mentioned so far is that we are using the same bond dimension everywhere which does not need to be the case

To do

- Include regularization schemes

Chapter 5

Results

5.1 Model

In this chapter we want to discuss the results for the ground state search using the modified NQS ansatzs that we have introduced. We compare the results obtained with the original NQS states. The model that we study is the Transverse Field Ising Model (TFIM) with Open Boundary Conditioins (OBC), the hamiltonian of which is given as follows

$$\hat{H} = -\sum_{l=0}^{L-2} \hat{\sigma}_l^z \hat{\sigma}_{l+1}^z - g \sum_{l=0}^{L-1} \hat{\sigma}_l^x \tag{5.1}$$

where σ_l^z and σ_l^x are the Pauli matrices, g is the external magnetic field and L is the length of the spin chain.

We choose TFIM because it is the easiest and the basic spin model and one can obtain an exact solution using JW transformation so that we can do a benchmark with the exact solution. We have studied the model for three different g namely below the critical point (h = 0.7), at the critical point (h = 1.0) and above the critical point (h = 1.3). We test our model on both real and complex parameters with different optimization algorithms , vanilla GD and SR.

First we discuss the case of complex parameters with SR as this is the case which we expect to find the ground state efficiently since the Hilber Space is complex. In figure 5.1 we show the accuracy of the new NQS state, quantified by the relative error on the ground-state energy $\epsilon_{\text{rel}} = \left(E_{\text{NQS}^{\text{new}}}(\alpha/D) - E_{\text{exact}}\right)/|E_{\text{exact}}|$, for different values of possible Bond Dimensions, D. Once we decide which kind of decomposition to an MPO we want to carry out, we have a list of possible Bond Dimensions that we can choose for this MPO. We fix $\alpha = 1$ and study the accuracy of a particular decomposition for different possible bind dimensions, D. For example for 16 spins we can do the decomposition as shown in figure [reference]. Given a particlular decomposition, we can vary D from minimum (D = 1) to the maximum possible value governed by the SVD. We have plotted 10 different initializations for each bond dimension D. This is done by choosing a different seed for each initialization. As we can see some of these initializations produce good accuracy and some of them do not. Also we plot the medians of these different initializaiton results.

Now let us consider the specific case when we do a decomposition into [2, 8] which puts the upper bound as $D < 4$.

In the plot we observe that the new NQS ansatz with $D = D_{\max} = 4$ produces approximately as good accuracy as the original NQS ansatz which includes a full weight matrix, W instead of an MPO. This is an expected resut since both the models have almost the same number of variational parameters so the new ansatz should in principle at least do as good as the original NQS ansatz. Also as we increase D we get a better accuracy which is also an expected since we are increasing N_{var}. What is interesting to notice is that the W matrix itself does not seem to have a particular structure since after performing SVD and calculating the energy with a truncated W matrix, the accuracy drops quickly. This simply show that W does not have a certain structure and hence is not compressible. More interestingly what we observe is that if we initialize our network with the new ansatz i.e., replacing the weight matrix W with an MPO, we can obtain a good compression in N_{var} without loosing accuracy dramatically. The $D = 2$ is doing as good a job as the full matrix W.

We can see a similar behaviour when we chose the [4, 4] decomposition where we have $D \leq 16$. For example for $D = 4$ and $D = 5$ we have some data points which are almost doing as good as the full W matrix but at the same time have less N_{var}.

5.1.1 16 spins data

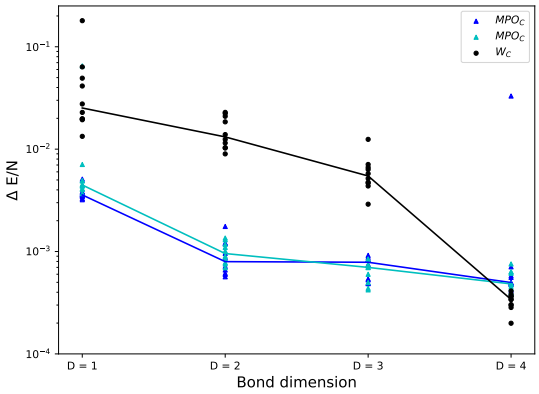


Figure 5.1: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 16 spins TFIM with OBC. The MPO decomposition used is [2, 8] for the cyan and [8, 2] for the blue color.

5.1.2 32 spin data

Below we plot some data with 32 spins

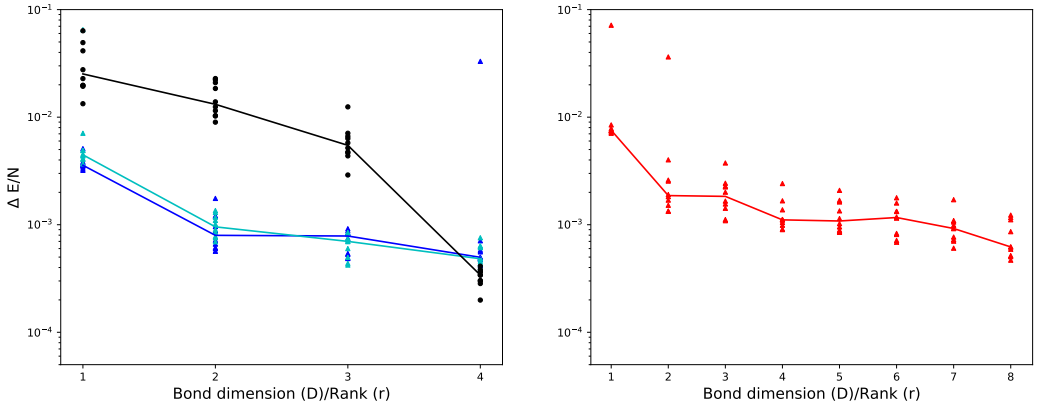


Figure 5.2: Left: [8, 2] decomposition. Right: [4, 2, 2] decomposition with $D_1 \leq 8$ and $D_2 = 2$

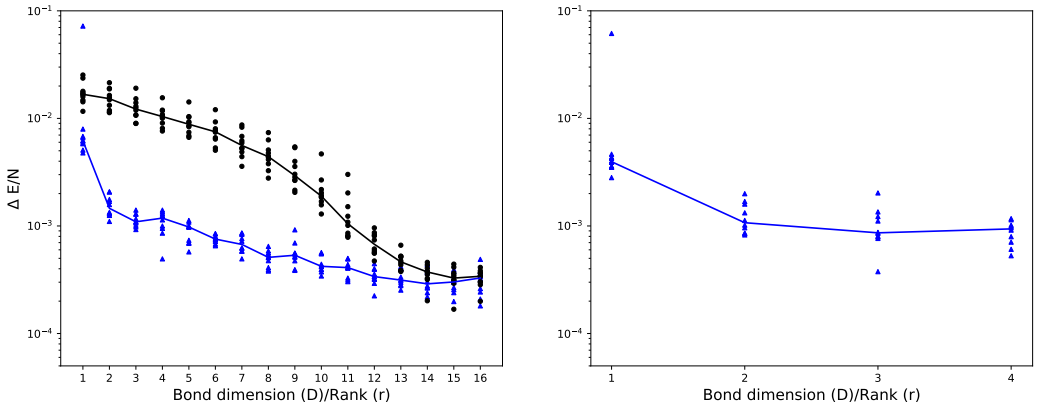


Figure 5.3: Left: [4, 4] decomposition. Right: [4, 2, 2] decomposition with $D_1 = 5$ and $D_2 \leq 4$

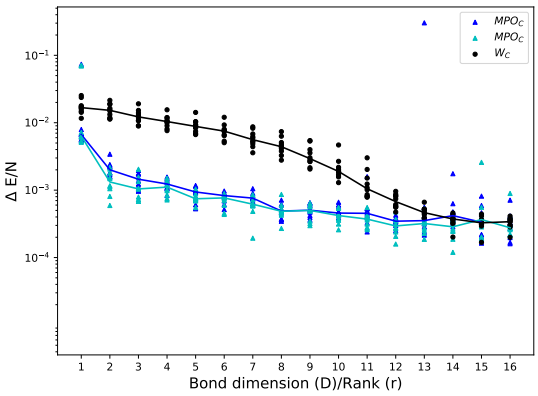


Figure 5.4: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 16 spins TFIM with OBC. The MPO decomposition used is [2,2, 4] for the cyan and [4, 2, 2] for the blue color.

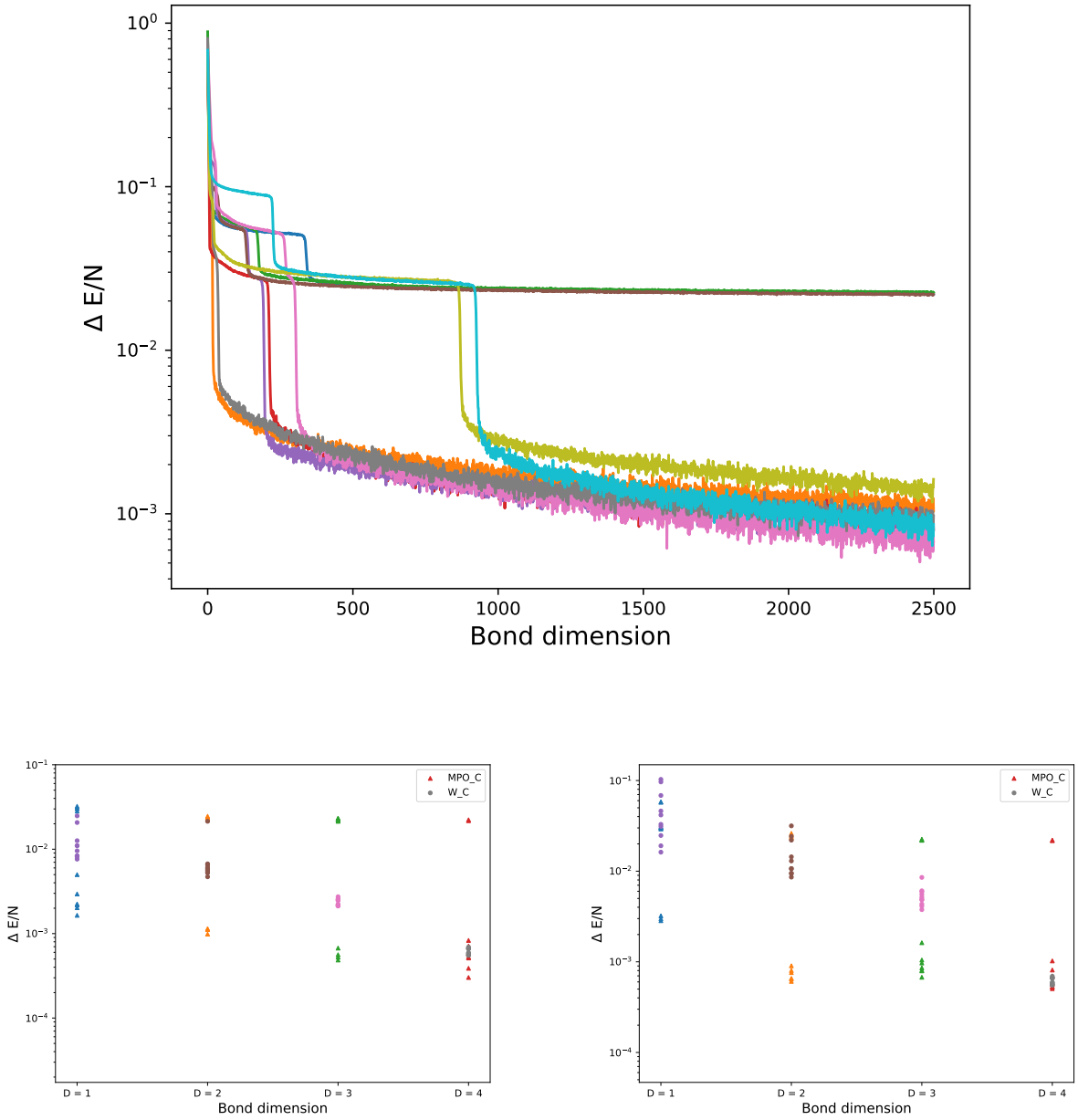


Figure 5.5: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. (Left) 2 node MPO with the decomposition [2, 16] in the input and the output dimensions of the legs. (Right) 2 node MPO with the decomposition [16, 2] in the input and the output dimensions of the legs

5.1.3 64 spins data

5.2 Comments on plots

- Different sizes with similar bond dimensions can be plotted on the same plot
- Plot different values of h on the same plot.

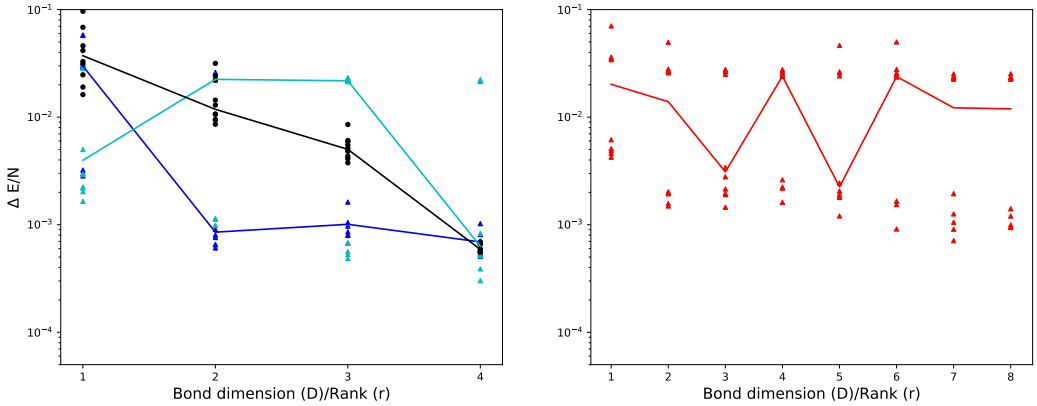


Figure 5.6: Left: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. The black blobs represent the data points for compressed W matrix via SVD, the cyan represent [2, 16] decomposition and the blue ones represent [16, 2] decomposition. Right: Decomposition of the blue cuve [16, 2] $\rightarrow$[4, 2, 2] with $D_1 \leq 8$ and $D_2 = 2$

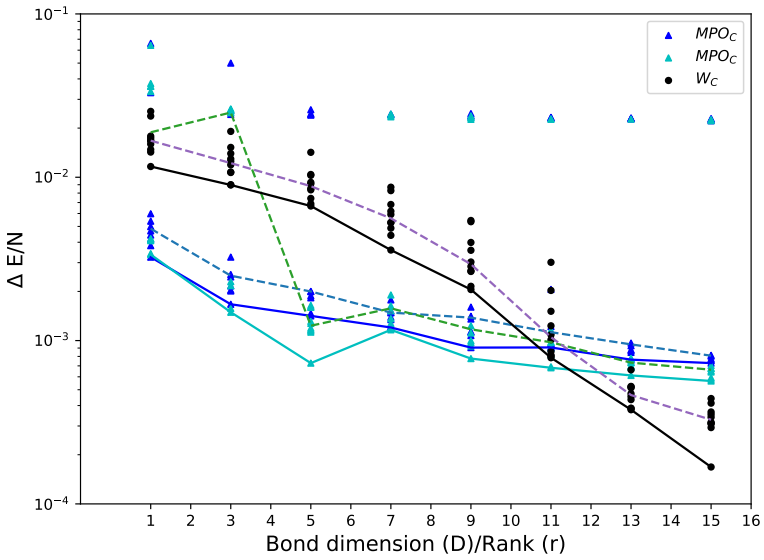


Figure 5.7: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. The black blobs represent the data points for compressed W matrix via SVD, the blue ones represent [8, 4] and the cyan ones represent [4, 8] decomposition. – $\rightarrow$ median could be a step size problem or just that [small, big] behaviour

- Compress plots along the x axis.
- Plot the minima with a solid like and the median with a dashed line
- What can be said about the outliers getting stuck at the same point for different bond dimensions for example in [4, 8] and the [8, 4] decompositions

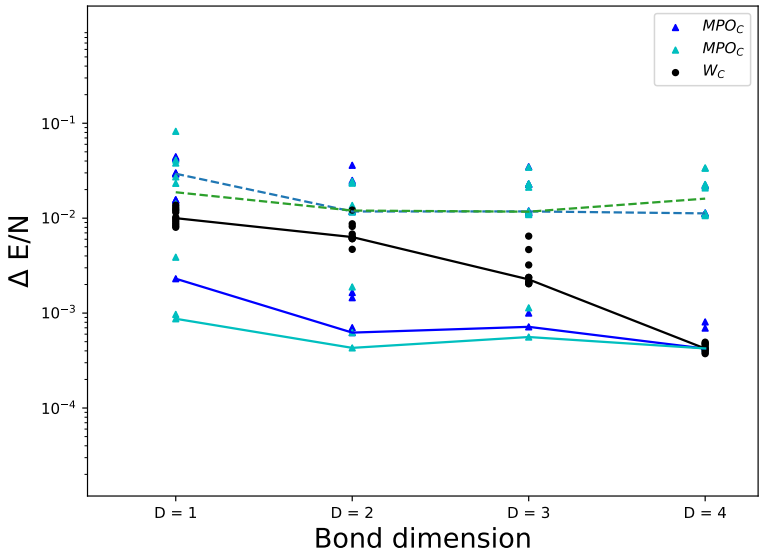


Figure 5.8: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 64 spins TFIM with OBC. The black blobs represent the data points for compressed W matrix via SVD, the blue ones represent [32, 2] and the cyan ones represent the [2. 32] decomposition.

5.3 Pending decompositions

- red → Finished, green → Running, blue → To be run
- 16 spins
  - compress [4, 4] → [2, 2, 4] with $D_1 \leq 5$, and $D_2 = 4$ Done! unplotted (exp. to be same as [4,4,2])
  - compress [8, 2] → [4, 2, 2] with $D_1 \leq 8$ and $D_2 = 2$. Done! plotted
- 32 spins
  - simulate [4, 8] with $D \leq 4$ and the possible compression
  - simulate [8, 4] with $D \leq 4$ and the possible compression
  - compressing [16, 2] → [8, 2, 2] with $D_2 = 2$ and $D_1 \leq 8$ done! plotted in figure 5.6
- 64 spins
  - simulate[2, 32] done! unplotted and the possible compression
  - simulate [32, 2]with $D \leq 4$, and the possible compression
  - simulate [8, 8]with $D \leq 64$, and the possible compression

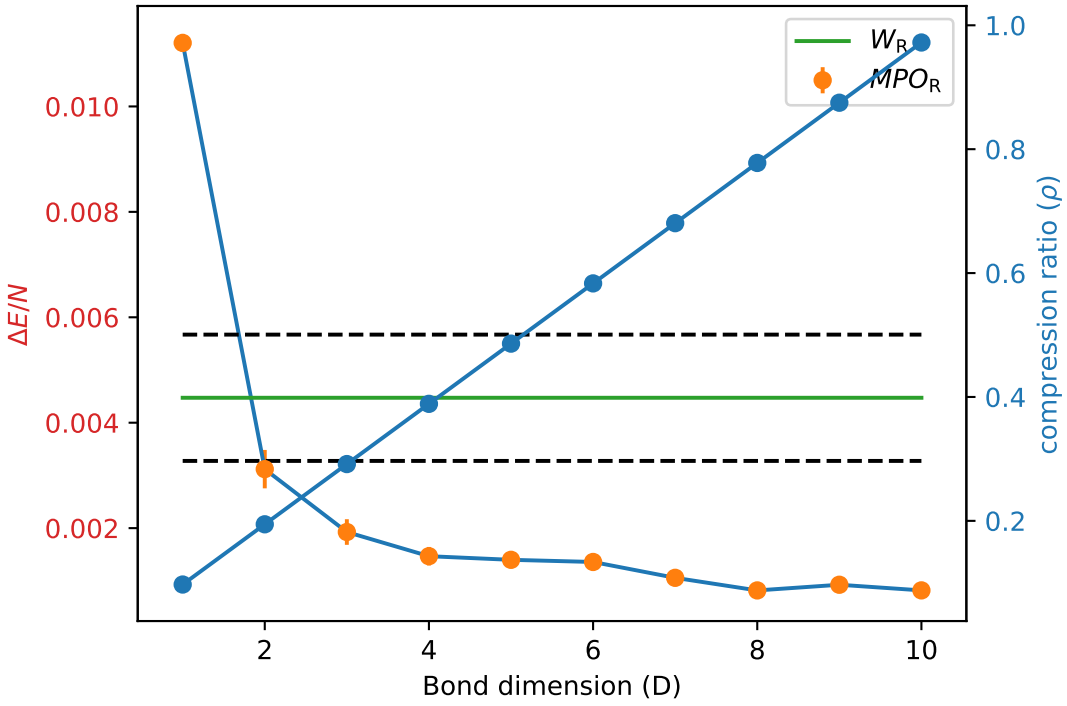


Figure 5.9: Plot for 12 spins with $\alpha = 3$ and [4, 3] decompotisiotn for Simple Gradient Descent

Chapter 6

Conclusion and Outlook

6.1 Conclusion

- Hybrid architectures of this kind do work, both with real and complex parameters.
- W matrix does not have any particular structure.
- We found that corresponding to a W matrix we can introduce an MPO.
- We see that using the MPO, a compression in N_{var} is possible.
- Different decompositions have different optimization behaviour.
- *Gradient descent* gets improved when used with real parameters.
- This approach however restricts us about which system sizes can be studied.

6.2 Outlook

- Compression can help to simulate larger quantum systems compared to the original RBM.
- This form of hybrid modelling can be used in any architecture in general where we have a matrix with large number of parameters irrespective of whethere we are dealing with the quantum systems or classical data.
- Although we could not explore it more but it would be interesting to know if different decompositions carry infomration about the entanglement in the system.

Chapter 7

Appendix

7.0.1 Appendix A.1: Proper gradient initialisation Already included in the main text

Numerical methods in general and machine learning in particular plagues with th eproblem that if we are start with a bad initial condition (guess of the wave-function), one can face problems like *Vanishing gradient/Barren plateau* problem. In this case what happens is that one is starting with a initial condition where the gradients of the wavefunctino with respect to the variational parameters of the model are vanishing. This essentially means that the model is not able to update its parameters and hence we are not moving anywhere is the energy landscape.

While we did not face such proble when dealing with the real parameters but we did face such issues when we started incorporating the MPO isnide our model with complex variational parameters. To get rid of this issue we followed the idea of calculating the variances of the gradients and initializing the network in such a way that the gradients of the wave function in our network do not have a 0 variance. Having a non zero variance in our gradients in makes sure that we have some fluatuatioins in our model and hence we are able to move down to the plateau where we face the vanishing gradient problem.

In the following we consider the model where we have included two nodes, inside the architecture. Later we will generalise it to n nodes.

$$Var\left[O_k^\theta(\mathbf{s})\right] = Var\left[\partial_{\theta_k} \log \psi_\theta(\mathbf{s})\right] \tag{7.1}$$

$$\psi_\theta(\mathbf{s}) = e^{\sum_{i=1}^N a_i s_i} \prod_{i_1=1}^{L_1} \prod_{i_2=1}^{L_2} .... \prod_{i_n=1}^{M/L_1 L_2 ... L_{n-1}} 2\cosh\left(\chi_{i_1 i_2 .... i_n}\right) \tag{7.2}$$

$$\log \psi_\theta(\mathbf{s}) = \sum_{i=1}^N a_i s_i + \log(2) + \sum_{i_1=1, i_2=1 ...., i_n} \log\left(\cosh\left(\chi_{i_1, i_2 ... i_n}\right)\right) \tag{7.3}$$

Assuming $a_i = 0$

$$\frac{\partial \log \psi_\theta(\mathbf{s})}{\partial A_{kp}^q} = \sum_{i_1=1, i_2=1 ...., i_n} \left(\frac{\sinh\left(\chi_{i_1, i_2 ... i_n}\right)}{\cosh\left(\chi_{i_1, i_2 ... i_n}\right)}\right) \frac{\partial\left(\chi_{i_1, i_2 ... i_n}\right)}{\partial A_{kp}^q} \tag{7.4}$$

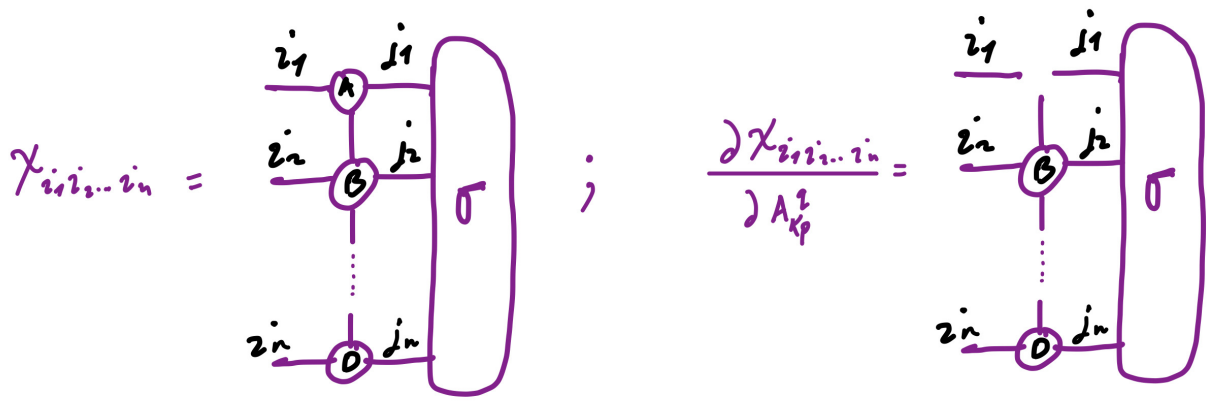


Figure 7.1: On the left we have the contraction of the reshaped input configuration with the MPO with "n" number of nodes $\chi_{i_1,i_2,..,i_n}$ and on the right we have a derrivative of this network with respect to the elements of a prticular node (here w.r.t the elements of the node A i.e., $\partial(\chi_{i_1,i_2...i_n})/\partial(A^q_{kp})$

.

Now from the law of large random numbers we know that the variance of the sum of random variables is simply equal to the sum of the variance of the random variables. Also if the mean of the random number is zero (which is the case for us) then one can also write the variance of the product of the random variables as the product of the variances of the random variables. We can ignore the variance of the input data since it is one In addition we make the appoximation for the $\tan(\chi) \simeq \chi$. Using these approximations we can get to the follwoing result:

$$V_k = \begin{cases} \frac{1}{n_{jk}} \left(\frac{\prod_{m=1}^{n} n_{im} v}{n_k^{2n-1} n_D} \right)^{1/(2n-1)} & ; \text{If } k = 1 \text{ or } k = n \\ \frac{1}{n_{jk}} \left(\frac{\prod_{m=1}^{n} n_{im} v}{n_k^{2n-1} n_0^{2n}} \right)^{1/2n-1)} & ; \text{otherwise} \end{cases}$$

where V_k is the variance with which we should initialize the k_th node of MPO so that our gradients have a variance of v. $n_{i_1}, n_{i_2}, ..., n_{i_n}$ are the dimensions of the MPO legs labelled with $i_1, i_2, ...i_n$ and similarly $n_{j_1}, n_{j_2}, ..., n_{j_n}$ are the dimensions of the MPO legs labelled with $j_1, j_2, ...j_n$. n_D is the bond dimension of the MPO which we have assumed to be the same throughout.

$$Var\left[\frac{\partial \log \psi_\theta(\mathbf{s})}{\partial A^q_{kp}}\right] = \tag{7.5}$$

- Automatic differentiation

Bibliography

[1] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science (80-.).*, 355(6325):602–606, 2017.

[3] Yusuke Nomura. Helping restricted Boltzmann machines with quantum-state representation by restoring symmetry. *J. Phys. Condens. Matter*, 33(17), 2021.

[4] Ze Feng Gao, Song Cheng, Rong Qiang He, Z. Y. Xie, Hui Hai Zhao, Zhong Yi Lu, and Tao Xiang. Compressing deep neural networks by matrix product operators. *Phys. Rev. Res.*, 2(2):23300, 2020.

[5] Markus Schmitt and Markus Heyl. Quantum Many-Body Dynamics in Two Dimensions with Artificial Neural Networks. *Phys. Rev. Lett.*, 125(10), 2020.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *J. Mach. Learn. Res.*, 9:249–256, 2010.