Universität Bonn

Compressing Neural Quantum States using Matrix Product Operators

Author:
Junaid Akhter
January 26, 2022

Supervisor:
Professor Matteo Rizzi

Contents

Chapter 1

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 2

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.

2.1 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:

$$|\Psi\rangle = \sum_{s_1, s_2 \cdots, s_N = 1}^{p} \psi\left(s_1, s_2 \cdots, s_N\right) |s_1\rangle \otimes |s_2\rangle \otimes \cdots \otimes |s_N\rangle \tag{2.1}$$

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

2.2 Neural Quantum States (NQS)

After their introduction NQS have found applications in studying a variety of peroblems. They have been used to study both quantum spin systems without geometrical frustation [10 from [3]] 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]]. Since the techniques in NQS based optimization are derrived from Machine Larning, we are going to dive in a gentle introduction to Machine Learning and Deep Neural Networks (DNN) in the follwoing sectiions.

2.2.1 Basics of Machine Learning (ML)

There is no exagerration in saying that Machine Learning (ML) is one of the most exciting and dynamic areas of modern research and application. One reason for that is that in the last three decades, we have seen a tremendous increase in our ability to generate and analyze large data sets. Also during the past few years ML has grown rapidly as interdisciplinary field. The techniques from machine learning are being employed in many scientific disciplines. [18-20 QNNS:review] including physics.

In this section we are going to go through a brief introduction to Machine Learning. In a typical Machine learning setting one starts with a dataset $D = (\mathbf{X}, \mathbf{y})$ where $\mathbf{X}$ is a matrix of independent variables and $\mathbf{y}$ is a vector of dependent variables depending on $\mathbf{X}$. One then needs a model $f(\mathbf{x}; \theta)$, which is a function $f : \mathbf{x} \rightarrow y$ of the parameters θ. To measure how well the model performs on the observations $\mathbf{y}$ a cost function $C\big(\mathbf{y}, f(\mathbf{X}; \theta)\big)$ is defined[Ref: Marin's paper]. The process of learning is finding the right parameters θ which minimize the loss function which for example could be takes as the squared error. This can be achieved by using the *gradient descent algorithm* (GD) where the variational parameters are updated iteratively to lead to wards the minimum minimum of the cost function in the parameters space. Intuitively one can think of the cost function as a landscape (figure 2.1) where a particular point is marked by the model parameters. The goal of learning/optimization is to go down this landscape until we reach the lowest point. In this way the model has successfully learned the network parameters such that we can get good accuracy on the unseen data. Formally what happends is that after initializing the network parameters say θ, the gradient descent algorithm updates these parameters by calculating the gradients $\nabla C = \left(\frac{\partial C}{\theta}\right)$ at each step of optimization in the follwing manner.

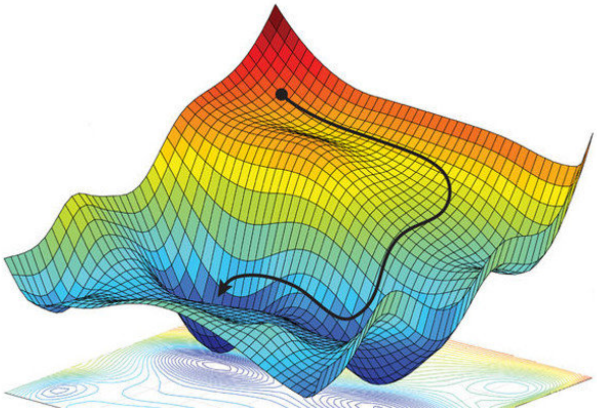


Figure 2.1: source of image

$$\theta \rightarrow \theta' = \theta - \eta \frac{\partial C}{\partial \theta} \tag{2.2}$$

where η is a hyperparameter of the model known as the learning rate. However, in practice another variant of GD called *stochastic gradient descent* (SGD) [reference] is used. This is due to the reason that in GD we have to go through all the samples from the data set in order to perform an update according to the equation 2.2 compared to the SGD where random samples from the data set are used to perform the trainig of the model.

The form of ML that we discussed above where we had access to dependent variables (also known as lables in ML community), $\mathbf{y}$ is known as *supervised machine learning.* One carries out the training of the model on a given set of data and after learning the model parameters applies the model to classify unknown data. A very famous example would be the classification of the hand written digits [referebce] or the fashion MNIST data [reference]. In physics this supervised learning has found applications in identifying the different phases of matter from the experimental data [reference]. *Unsupervised machine learning* is another form of learning where we do not need the pre assigned labels for the data. Depending upon the nature of the problem one has to carefully choose the loss function which one wants to minimize. This form of learning is used for example for clustering, dimensionality reduction etc. Finally we have *reinforcement learning* in which there is an agent which takes actions in the environment and takes some decisions depending upon the policy based on the obervation made on the environment. Reinforcement learning has applications in a variety of fields like game throry, control theory, operations research, information theory etc.

Given the setting for learning the ground state search using NQS, we can say that this methods belongs somewhere in between the supervised learning and the reinforcement learning. This will become more clear as we follow along.

- definition and structure, different forms of learning
  - Supervised learning: Where the model is trained over some training data set which is labelled. Classification is one application of supervised learning.
  - Unsupervised learning: In this learning learning is done on the data where there are no labels. This is mainly used for clustering, dimensioinality reduction etc.
  - Unsupervised learning: This can be called the most complicated form of learning where there is an agent which interacts with the environment and takes decisions accordingly.
- DNN as universal function approximatrors
- Different types of Neural networks

2.2.2 Machine Learning (DL)

Deep Learning (DL) is a subset of Machine Learning (ML) which is one of the most exciting and dynamic areas of modern research and application. In this section we first want to give an introductory account of the ingredients of ML and then we will constraint ourselves to DL. 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 assigns the images to if it is 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/ML referes to finding the right set of parameters $\boldsymbol{\theta}^*$ such that we fit the data well, i.e.,

$$f(\mathbf{x_n}, \boldsymbol{\theta^*}) \approx y_n \qquad \text{for all } n = 1, ..., N. \tag{2.3}$$

In other words what this means is that our predictor has learnt the features of the input data $\mathbf{x}$ (cats and dogs). After learning if we give an unseen (images other than used in training) set of images to our predictor, it should be able to classify them with good acuracy. In general following three ingredients are necessary to perform such a task [reference UC Berkeley]:

- *A Decision Process/Model:* A decision process/model is needed to make a prediction or classification of the input data. In *supervised learning* the data is labelled but in general this is not necessary in other forms of 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 partucular prediction. In the example given above, it will tell us how many times have assigned a picture to a wrong label i.e., the model identifies a dog as a cat or vice versa.

- *An Optimization Process*: Given that we have a measure of how well our model is fitting the data through the loss function l, we want to iteratively optimize our parameters $\boldsymbol{\theta}$ in such a way so that we minimize l. This is done with an optimization method like *Gradient descent* or its other variants [reference].

So far we have considered the predictor $f(\mathbf{x_n}, \boldsymbol{\theta}^*)$ as a general function. Deep Learning refers to the form of learning where our predictor/model is an *Aritificial Neural Network* which discussed below:

It is a usual practice to divide the dataset into learning D_{learning} and D_{test}.

Artificial Neural Networks

One reason for that is that in the last three decades, we have seen a tremendous increase in our ability to generate and analyze large data sets. Also during the past few years ML has grown rapidly as interdisciplinary field. The techniques from machine learning are being employed in many scientific disciplines. [18-20 QNNS:review] including physics.

2.2.3 Artificial Neutal Networks (ANN)

Artificial neural networks (NN) are one of the most important structures which are used in ML algorithms. As the name suggests ANN are the networks which are supposed to mimic the behaviour of human brain so that computer programs can perform tasks like pattern recognition and other forms of problem solving in the field of AI and ML.

Formally, a neural net is made up of basic units called neurons, i 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 $\sigma_i(z)$ (figure 2.2).

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)}) = \sigma_i(\boldsymbol{w}^{(i)}.\mathbf{x} + b^{(i)}) \tag{2.4}$$

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 the weights and biases, $\mathbf{w}^{(i)} = (b^{(i)}, \boldsymbol{w}^{(i)})$ are the variational parameters in ML models. 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, rectified linear units (ReLUs), leaky rectified linear units (leaky ReLUs), and exponential linear units (ELUs) (figure 2.3). Depending upon the choice of the non-linearity one expects differen computational and training properties of neurons.What are the restrictions on the non-linearities to be used?

A NN can be constructed either with just one layer of neurons or stack different layers together with the continuity that the output of one layer serves as an input to the next layer in which case it will be called a *deep neural network* (DNN). 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 functioin with arbitrary accuracy[reference

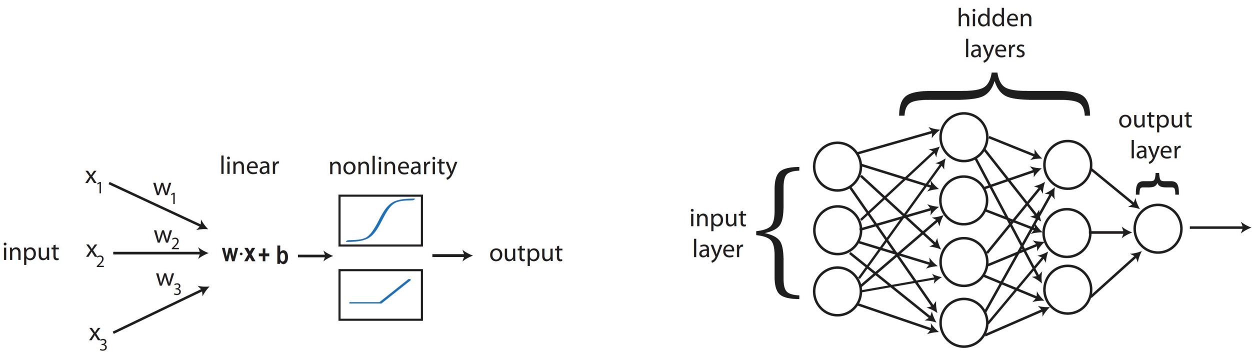


Figure 2.2: (left) The basis components of a neural network which consists of a linear transformation and followed by a non-linear transformation; (right) Neurons are stacked into layers with the output of one layer serving as the input to the next layer.[Reference from Marin]

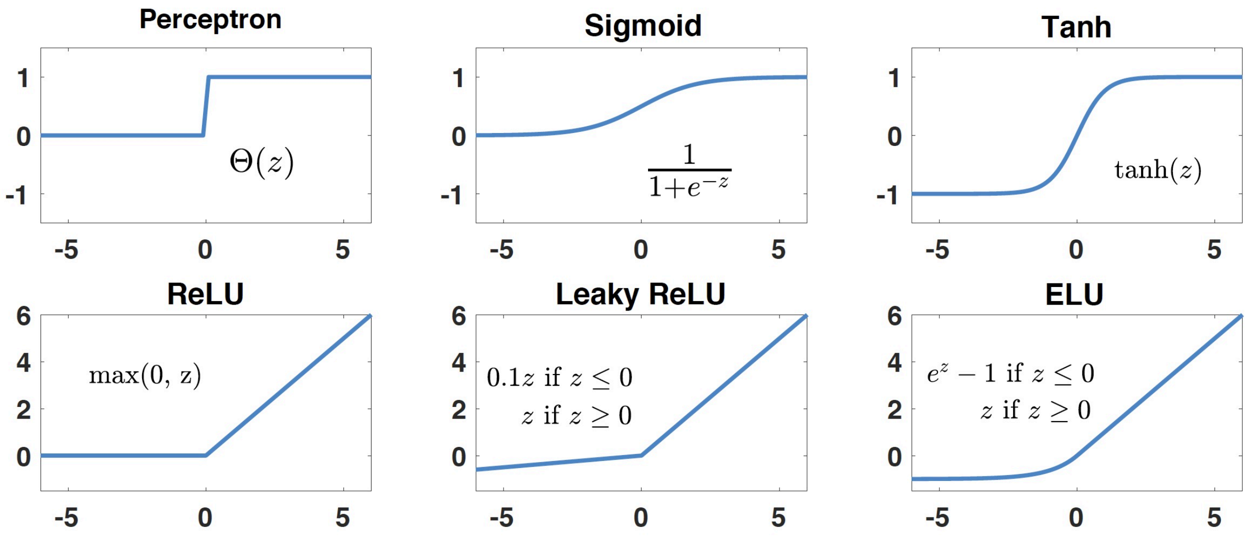


Figure 2.3: Different activation functions [Marin]

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. Depending upon the nature of data and the task that on wants to achieve a suitable architecture is chosen. For example if one wants to classify hand-written digits from the MNIST data-set then the suitable architecture for this task would be the CNNs. RBM are mostly used in unsupervised learning to performs clustering or dimensionality reduction.
learning etc.

The property that DNN are universal function approximators have made them mainstream in machine learning community. This is also a reason that one can expect them to represent QMBWF. In this work we are going to work with RBM as was also done in the original work [2]. In the following section we introduce RBM and try to understand its structure from a *graph theoretic* point of view.

Restricted Boltzmann machine (RBM)

RBM is a special case of the *Boltzmann machine* (BM) also known as the *stochastic Hopfield network* with hidden units. BM belong to the class of neural networks which are called energy-based neural network models [53, 54 in review]. Compared to neural networks used for supervised learning which were discussed in the previous section, BM is different in the sense that there is no activation function attached to each specific neuron. In other words rather than treating each neuron individually, BM treats neurons as a whole.

A Boltzann machine can be regarded as particular graphical model [22 in notes] and is very similar to the classical Ising model. It is customary to desceibe BM into the framework of probabilistic graphical models since it provides an immediate acess to well established algorithms and wealth of theoretical results. An *undirected graph* is a tuple G = (V, E) , where V is a finite set of nodes and E is a set of undirected edges. An edge consists out of a pair of nodes from V. Now, given a graph with edge set *E(G)* and the vertex set *V(G)* the neurons $s_1, s_2, \cdots, s_n$ (equivalent to spins in the Ising model) are put on the vertices, $n = |V(G)|$. A weight elemtent, w_{ij} is associated if the two vertices i and j are connected. The neurons $s_1, s_2, \cdots, s_n$ are equivalent to the spins and the wright elements w_{ij} are the equivalent to the coupling constants in the Ising model. Also for each neuron s_i, there is a corresponding local bias b_i which corresponds to the local field in the ising model. Similar to the Ising model, for each series of input values/configurations $s = (s_1, s_2, \cdots, s_n)$, energy/hamiltonian is defined as

$$E(\mathbf{s}) = -\sum_{\langle ij\rangle \in E(G)} w_{ij} s_i s_j - \sum_i s_i b_i \tag{2.5}$$

$$\tag{2.6}$$

So far everything is the same as know in the Ising model. The main difference is a constraint on the BM which is that each vertex receives a lable *hidden* or *visible.* So what we have is k hidden neurons, $h_1, h_2, \cdots, h_k$, and l visible neurons $v_1, v_2, \cdots, v_l$ such that $k + l = n$. In this way we have parametrized the energy as E $(\mathbf{h},\ \mathbf{v})$. We define BM as a parametric model of a joint probability distribution between variables $\mathbf{h}$ and $\mathbf{v}$ with the probability given by the Gibbs distribution

$$p(\mathbf{h}, \mathbf{v}) = \frac{e^{-E(\mathbf{h},\mathbf{v})}}{Z} \tag{2.7}$$

where $Z = \sum_{\mathbf{h},\mathbf{v}} e^{-E(\mathbf{h},\mathbf{v})}$ is the usual partition function. It is precisely because of equation 2.6 that such models are called energy based models and because of equation 2.7 that the name is a Boltzmann machine.

It turns out that the learning process wth the general BM is difficult and computationally expensive. This training can be made easier by imposing some restrictions on the network topology. This leads to an RBM where the graph G is a bipartite graph (see figure 2.4). The network consists of N visible units $\mathbf{V} = (v_1, v_2, \cdots v_N)$ and M hidden units $\mathbf{H} = (h_1, h_2, \cdots h_N)$ with the constraint that there is no interaction between the hidden and the visible units . The energy as a function of hidden and visible units can be written as:

$$E(\boldsymbol{v},\boldsymbol{h}) = -\sum_{i=1}^{N}\sum_{j=1}^{M} w_{ij}h_i v_j - \sum_{j=1}^{M} b_j v_j - \sum_{i=1}^{N} c_i h_i \tag{2.8}$$

It has been shown that this kind of restricted architecture can approximate every discrete probability distributioni [63, 64 from review] and hence been extensively investigated and used [26-29, 31, 38, 39, 55-60 in review.] In the next section we are going to see how RBM can be used to endode the QMBWF.

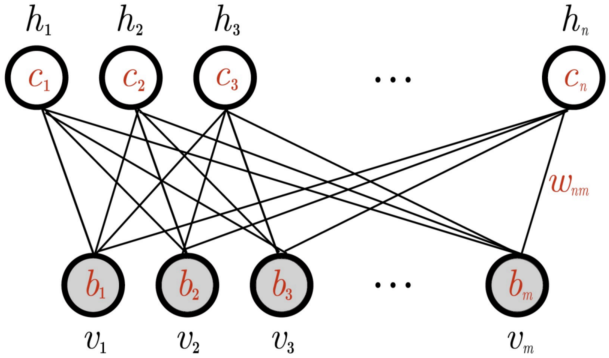


Figure 2.4: Interaction between hidden and visible units [notes]

- Def: parametrization of a multilayer mapping of signals in terms of many alternatively arranged linear and non-linear transformations
- Universal function approximators
- Role in Machine Learning and Physics
- How the parameters are updated and how the learning takes place
- Restricted Boltzmann Machines

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

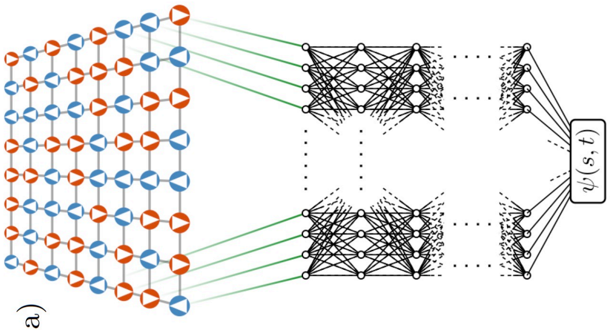


Figure 2.5: NQS representation of a many body wave-function

- Will represent QMBWF as RBM as discussed earlier
- write the variational form of the wave function
- Introduce RBM for wave function
- write the simplified version
- write how the gradients look
- conclude that this used as a variational ansatz for VMC

Given that NN are universal function approximators, an interesting question to ask is that can NN represent QMBWF. It turns out that they do and in principle one should be able to use any architecture to represent a QMBWF given that it has sufficient amount of neurons in it. However, as mentioned earlier in this work we will use RBM as our ansatz for representing the wave-function as was also used in the original work by [2]. The architecture is defined such that the states of the physicsl degrees of freedom (e.g., spins σ_i^z in the computational basis) identify with those of the visible units $v_1, v_2, \cdots, v_{\mathrm{N}}$ (N is the system size) while as the the hidden units $h_1, h_2, \cdots, h_{\mathrm{M}}$ capture the dependencies between the observed/visible variables (check figure 2.6).

$$\psi_{\boldsymbol{\theta}}(\mathbf{s}) = \sum_{\{h_i\}} e^{\sum_{ij} W_{ij} h_i \sigma_j^z + \sum_j a_j \sigma_j^z + \sum_i b_i h_i} \tag{2.9}$$

where $h_i = \{-1, 1\}$ is a set of M hidden spin variables, and the wrights, $\boldsymbol{\theta} = \{a_i, b_i, W_{ij}\}$ are the variatiional parameters of the network which determine to output of the network when given a particular input spin configuration, $\mathbf{s}$ as shown in figure 2.5.

In equation 2.9 the wave-function is interpretted as the generalised probability extended to complex numbers. Since in our case our observables are spins, our visible units are spin states in the computational basis and hence equivalent in number to the system size N.

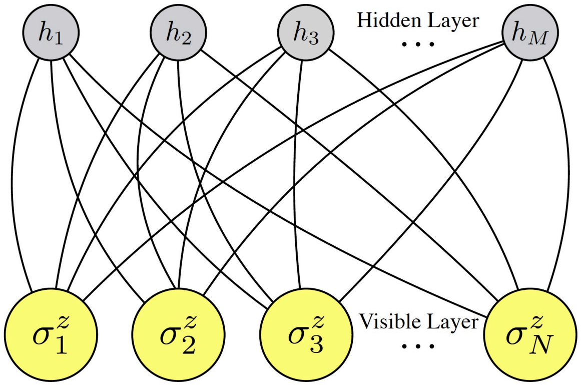


Figure 2.6: (left) The basis components of a neural network which consists of a linear transformation and followed by a non-linear transformation; (right) Neurons are stacked into layers with the output of one layer serving as the input to the next layer.[Carleo and Troyer]

For each input configuration in equation 2.1 we want our network to map it to a complex number $\Psi(\mathbf{s}, \boldsymbol{\theta})$ which depends on both, the input configuration $\mathbf{s}$ and the variational parameters $\boldsymbol{\theta}$. In this respect we can write the variational state in the follwing manner.

$$|\Psi(\boldsymbol{\theta})\rangle = \sum_{\mathbf{s}\in\mathbb{Z}_p^N} \Psi(\mathbf{s}, \boldsymbol{\theta})|\mathbf{s}\rangle \tag{2.10}$$

where the sum runs over all basis lables, $|\mathbf{s}\rangle$ denotes $|s_1\rangle \otimes \cdots |s_N\rangle$ and $\mathbb{Z}_p$ the set $\{0, 1, \cdots, \text{p-1}\}$. Since there are no intra-layer interactions in an RBM, we can trace out the hidden units in equation 2.9 and write it in a more compact form

$$\psi_\theta(\mathbf{s}) = e^{\sum_i a_i s} \times \prod_{i=1}^{M} 2\cosh[b_i + \sum_j W_{ij} s_j] \tag{2.11}$$

Now that we have a useful form of the variational wave-function, we can use an optimization method like GD to optimize the variational parameters in order to update our wave-function for a particular task e.g., performing the ground state search. However, it has been found that GD performs poorly [reference] when used to optimize NQS. Instead what is used is *stochastic reconfiguration (SR)* which is discussed in the next section.

- NQS can encode Volume law states compared to Tensor Network states
- An introduction to RBM
- Write RBM Ansatz the Many Body Wave-function
  - Why this ansatz qualifies to be a representaiton of wave-function
- picture showing the interaction between the hidden and the visible layer
- Definition of the cost function in NQS

2.2.5 Varational quantum montecarlo

Varational Monte Carlo methods is a quantum MC method used for finding the ground state of quantum systems. VMC methods are based on the direct application of Monte-Carlo integration methods to QMBWF. The methods requires a trial wave-function also known as variational ansatz which is dependent on some adjustable parameters known as variational parameters. After the optimization of the trial wave-function we obtain the ground state wave-function of the system.
In our case the wave-function in equation 2.11 is the variational wave-function with θ as the variational parameters.
Given that we have acces to a variational wave-function, the expectation value of a local operator

$\hat{O}$ can be calculated as follows:

$$\begin{aligned}
\frac{\langle\psi(\theta)|\hat{O}|\psi(\theta)\rangle}{\langle\psi(\theta)\mid\psi(\theta)\rangle} &= \sum_{\mathbf{s},\mathbf{s}'}\frac{\psi_\theta(\mathbf{s})^*\psi_\theta\left(\mathbf{s}'\right)}{\langle\psi(\theta)\mid\psi(\theta)\rangle}\left\langle\mathbf{s}|\hat{O}|\mathbf{s}'\right\rangle && (2.12)\\
&= \sum_{\mathbf{s}}\frac{\left|\psi_\theta(s)\right|^2}{\langle\psi(\theta)\mid\psi(\theta)\rangle}\sum_{\mathbf{s}'}\left\langle\mathbf{s}|\hat{O}|\mathbf{s}'\right\rangle\frac{\psi_\theta\left(\mathbf{s}'\right)}{\psi_\theta(\mathbf{s})} && (2.13)\\
& && (2.14)
\end{aligned}$$

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

$$\frac{\langle\psi(\theta)|\hat{O}|\psi(\theta)\rangle}{\langle\psi(\theta)\mid\psi(\theta)\rangle} = \sum_{\mathbf{s}} p_\theta(\mathbf{s})O^\theta_{\mathrm{loc}}(\mathbf{s}) \equiv \left\langle\left\langle O^\theta_{\mathrm{loc}}\right\rangle\right\rangle_\theta \tag{2.15}$$

where we introduced the local estimator

$$O^\theta_{\mathrm{loc}}(\mathrm{s}) = \sum_{\mathrm{s}'}\left\langle\mathrm{s}|\hat{O}|\mathrm{s}'\right\rangle\frac{\psi_\theta\left(\mathrm{s}'\right)}{\psi_\theta(\mathrm{s})} \tag{2.16}$$

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 equation 2.14 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 $\{\ \mathbf{s}^{(1)}, \mathbf{s}^{(1)}, \ldots\mathbf{s}^{(N_{\mathrm{MC}})}\}_{\mathbf{s}\sim p_\theta((s))}$ and the expectation values can be approximated by the emperical mean as follows

$$\left\langle\left\langle O^\theta_{\mathrm{loc}}\right\rangle\right\rangle_\theta \approx \frac{1}{N_{\mathrm{MC}}}\sum_{n=1}^{N_{\mathrm{MC}}} O^\theta_{\mathrm{loc}}\left(\mathbf{s}^{(n)}\right) \tag{2.17}$$

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

Ground state search

The search for a ground state of a many body system means that we are looking for the state with minimum energy. This can be achieved by updating our wave-function in an iterative manner such that at each iteration we are moving towards a lower energy in the energy landscape as a function of $\boldsymbol{\theta}$. The energy expectation value calculated from the variatioinal ansatz 2.14 is given as follows:

$$E(\theta) = \frac{\langle \Psi(\theta)|\hat{\mathrm{H}}|\Psi(\theta)\rangle}{\langle \Psi(\theta)|\Psi(\theta)\rangle} \tag{2.18}$$

In order to move down towards the minima in the energy landscape using the GD algorithm, 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 equation 2.18

$$\nabla_{\theta_k} E(\theta) = 2\mathrm{Re}\left(\langle\langle (O_k^\theta)^* E_{loc}^\theta \rangle\rangle_\theta - \langle\langle (O_k^\theta)^* \rangle\rangle_\theta \langle\langle E_{loc}^\theta \rangle\rangle_\theta \right) \tag{2.19}$$

where $E_{loc}^\theta(\mathbf{s}) = \sum_s \langle \mathbf{s}|\hat{H}|\mathbf{s}'\rangle \frac{\psi_\theta(\mathbf{s}')}{\psi_\theta(\mathbf{s})}$ is defined as in 2.16 is the local Energy. Further we also introduced the logarithmic derivatives $O_k^\theta(\mathbf{s}) = \partial_{\theta_k} \mathrm{log}\psi_\theta(\mathbf{s})$ 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.

$$\theta_k^{(n+1)} = \theta_k^n - \tau \partial_{\theta_k} E(\theta)|_{\theta=\theta^n} \tag{2.20}$$

where τ is the learning rate. However, it has been observed that for spin Hamiltonians the energy landscape $E(\theta)$ is not convex which means that there is a possibiliity that the simple gradient descent optimization can get stuck in local minima and saddle points. To encounter this problem there are other optimization schemes available. Among them is the *Stochastic Reconfiguration* (SR) method [Reference] where the update the update step is done as follows:

$$\theta_k^{(n+1)} = \theta_k^n - \tau \mathcal{S}^{-1}_{k,k'} \partial_{\theta_{k'}} E(\theta)|_{\theta=\theta^n} \tag{2.21}$$

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]. It exploits the informatioin about the local geometry of the Hilbert Space and optimizes the variational parameters accordingly. Hence it is a better choice compared to the simple gradient descent. Another way of putting SR is that SR methods updates the variational parameters according to the GD weighed by the quantum Fisher matrix.

2.3 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 2.21 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.

- $$N_{var} \sim N_{site}^2$$
- 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 3

Compressing variational parameters inside NQS

As pointed out at the end of the previous chapter, our goal is to compress NQS so that we can do the optimization efficiently. We write a modified RBM ansatz for QBMWF which can give us a compressed NQS compared to the original RBM. The idea we use is to replace the full weight matrix W (which contains the most of the variational parameters (all in our case) with a *Matrix Product Operator* [4].

- Our goal is to use tools from Tensor Networks and Neural Networks and see if something useful can be created.
- 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.
- This could be a very useful for reasons 1). It is already promising for classical data for compression 2). It leads to a new direction of possibilities where NQS ansatz could be modified.

The hope is that the dimensionality reductioin property of *Tensor Networks* [reference](MPO is a tensor network) will enable us to write a compressed form of W as a multiplication of smaller sized matrices.

This kind compression has already been found useful in CNN for example where the data is classical (image pixels) and the task is the classification task.

The architecture is discussed in the next section in detail and is used to learn the ground state of *Transverse Field Ising Model*.

- It becomes difficult to manipulate the variational parameters which means larger systems can not be simulated easily.
- Compressing a DNN to reduce its N_{var} but not its prediction power is an important but chellanging problem

- Briefly discuss different possible compression strategies.

3.1 A brief intoduction to Matrix Product Operators

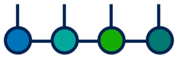
MPO are a class of *Tensor Networks (TN)* which were first introduced in physics to characterize the short-range entanglement in one-dimensional quantum many body systems [34-35 in [4]]. In mathematics it is known by the name tensor-train approximations [36-37 in [4]]. A TN is a factorization of a higher-order 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 $Ti_1, i_2, \cdots i_n$. The number of values that a pariticular index i_k can take is the dimension of i_k. A simplest tensor would be a number i.e., a tensor of rank 0. 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 (see figure). The graphical interpretation especially comes handy if we want to do a contraction between indices. For example We can obtain a new tensor $C_{i,j,m,n} = \sum$

The number of parameters in a TN scale lilnearly (polynomially in some cases) compared to the exponentially large hilbert space. This along with the ability to represent the low entanglement states of gapped hamiltonians make them very efficient in applications in condensed-matter physics [40,41 in [4]]. TN are also very intuitive when looked at diagramatically.

For a detailed discussion on the diagramatic notaion of tensor networks one can refer to [references]. In figure 3.1 we can see how a multidimensional tensor (Hilbert space) can be represented as a decompositioin into smaller tensors (bobs). The connected lines represent the contraction between these smaller tensors and the free lines represent the physical degrees of freedom. The dimension of the connecting/contracting edges is called the *bond dimension* and is usually denoted by D or χ. This is the variational parameters which can be varied and is selected in a way that we get a compressed wave-function and at the same time are also able to efficiently represent the wave-function without loosing the useful information. This is known as an MPS representation of the Hilbert Space and is the working horse for studying the One Dimensional Systems using DMRG. An operator which acts on this MPS is represented as Matrix Product Operator (MPO) which has extra free edges connected to these blobs as shown in figure 3.2.

In our modified NQS we exploit the idea that we can decompose the parameters matric W as a tensor-train. We see that using this representation we can gain compression in N_{var} without compromising over the accuracy with which the ground state is estimated.

Note: It is very important to note that in this work we are not representing our Hamiltonian by an MPO which acts on the MPS.



$$= \sum_{\alpha_1,\alpha_2,\alpha_3} A^{s_1}_{\alpha_1} B^{s_2}_{\alpha_1\alpha_2} C^{s_3}_{\alpha_2\alpha_3} D^{s_4}_{\alpha_3}$$

Figure 3.1: MPS

$$M^{s_1 s_2 s_3 s_4 s_5 s_6}_{s'_1 s'_2 s'_3 s'_4 s'_5 s'_6} = \sum_{\{\alpha\}} A^{s_1 \alpha_1}_{s'_1} A^{s_2 \alpha_2}_{\alpha_1 s'_2} A^{s_3 \alpha_3}_{\alpha_2 s'_3} A^{s_4 \alpha_4}_{\alpha_3 s'_4} A^{s_5 \alpha_5}_{\alpha_4 s'_5} A^{s_6}_{\alpha_5 s'_6}$$

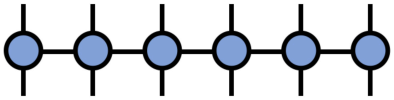


Figure 3.2: MPO

3.1.1 Interpretability

- Does the W matrix from RBM have any physical meaning and can MPO help us to understand it better in some way?
- Linearising data which has non-linearities by projecting to a larger state space and then drop the least important information (notes)
- Definition of SVD and how it can be used for data compression with pictorial explaination.
- Is it possible that we can compress W which means that it has a particular structure.
- Citing the plot that shows that this is not a useful strategy to do.
- SVD is only possible after training and hence will not reduce the training rime

3.2 Modified NQS ansatz

Using the idea that we just discussed above, we can write our modified NQS ansatz which is kind of a hybrid architecture between Neural Networks and Tensor Networks. So we replace the W matrix with an MPO with a fixed Bond dimension D. The dimensioin of the input legs of the MPO should be selected in such a way that it the overall dimension is equal to the number of the size of the system/input configuratioin. For dimension of external legs we make sure that we are comparing our MPO with an RBM with $\alpha = 1$. Both of these constraints can be satisfied in many choices. For example if we have a system size of 16 then the possible combinations for two nodes are $M^{2,8}_{2,8}$, $M^{8,2}_{8,2}$, $M^{4,4}_{4,4}$, for three nodes $M^{2,2,4}_{2,2,4}$, $M^{4,2,2}_{4,2,2}$ and for four nodes $M^{2,2,2,2}_{2,2,2,2}$ and the cross combinations. It also means that we want to only test the systems sizes which we can decompose into an MPO properly e.g., with a system size of 17 it will not be possible. For these reasons we find it convenient to simulate the systems which are powers of two. The number of variational parameters in the modified ansatz are now quadratic (for more than two nodes) in the bond dimension, D and given by the follwing formulla:

$$N_{\text{var}} \sim D^2 \tag{3.1}$$

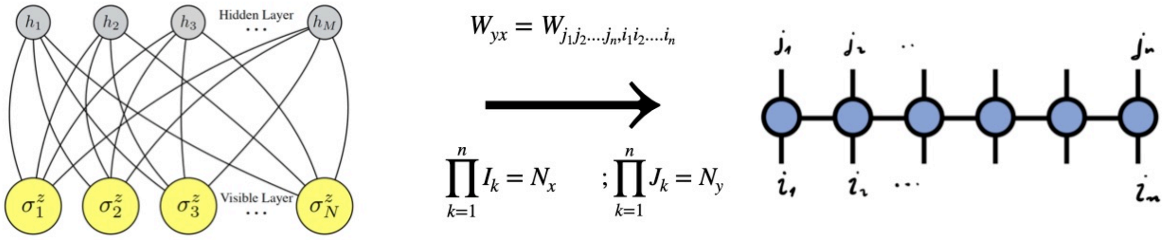


Figure 3.3: Replacement of the full matrix W with an MPOThe i and j indices need to be reversed.[create a better picture maybe]

The anstaz for the new wavefunction in which the full weight matrix, W has been replaced by an MPO with n nodes is given by 3.2 and pictorially by figure 3.3

$$\psi_\theta(\mathbf{s}) = e^{\sum_{i=1}^{N} a_i s_i} \prod_{i_1=1}^{L_1} \prod_{i_2=1}^{L_2} .... \prod_{i_n=1}^{M/L_1 L_2 \dots L_{n-1}} 2\cosh\left(\chi_{i_1 i_2 \dots . i_n}\right) \tag{3.2}$$

where

$$\chi_{i_1 i_2 \dots . i_n} = \sum_{\{\alpha\}} \sum_{j_1, j_2, .. j_n} \Lambda^{[1], i_1}_{j_1, \alpha_1} \Lambda^{[2], i_2}_{j_2, \alpha_1 \alpha_2} \dots \Lambda^{[n], i_n}_{j_n, \alpha_{n-1}} \sigma_{j_1, j_2 \dots j_n} \tag{3.3}$$

Λ^i denotes the i_{th} node of the MPO, $\{\alpha\}$ is the set of all the bond dimensions and $\sigma_{j_1, j_2 \dots j_n}$ is the reshaped input which is contracted with the MPO as shown in the figure 3.4. Equation 3.2 represents our new architecture/ansatz for the many body wave function. We initialize out network with this architecture and use our optimization scheme to get the ground state of the system. This is different that performing the SVD of the weight matrix at each iteration.
The logarithm of the wave-function need to calculate the expectation values are given by

$$\log \psi_\theta(\mathbf{s}) = \sum_{i=1}^{N} a_i s_i + \log(2) + \sum_{i_1=1, i_2=1 \dots ., i_n} \log\Bigg(\cosh\left(\chi_{i_1, i_2 \dots i_n}\right) \Bigg) \tag{3.4}$$

Now using 3.2 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 4.

- Pictorial and mathematical representaion of the new ansatz
  - Constrainst about which system sizes can be decomposed into an MPO
- 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

3.2.1 Proper initialization of the network

In order to apply the optimization procedure, we want to take the gradients of equation 3.4 with respect to the variational parameters which are the elements of the tensor nodes. Assuming $a_i = 0$

$$\frac{\partial \log \psi_\theta(\mathbf{s})}{\partial \Lambda^q_{kp}} = \sum_{i_1=1, i_2=1 \ldots, i_n} \left(\frac{\sinh\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\cosh\left(\chi_{i_1,i_2 \ldots i_n}\right)} \right) \frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda^q_{kp}} \tag{3.5}$$

$$= \sum_{i_1=1, i_2=1 \ldots, i_n} \tan\left(\chi_{i_1,i_2 \ldots i_n}\right) \frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda^q_{kp}} \tag{3.6}$$

In figure 3.4 we show the pictorial representation of the contraction of the input configuration with the MPO and also the tensor derrivative [reference for the tensor derrivative] with respect to the entries of the first node of the MPO i.e., $\frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda^q_{kp}}$

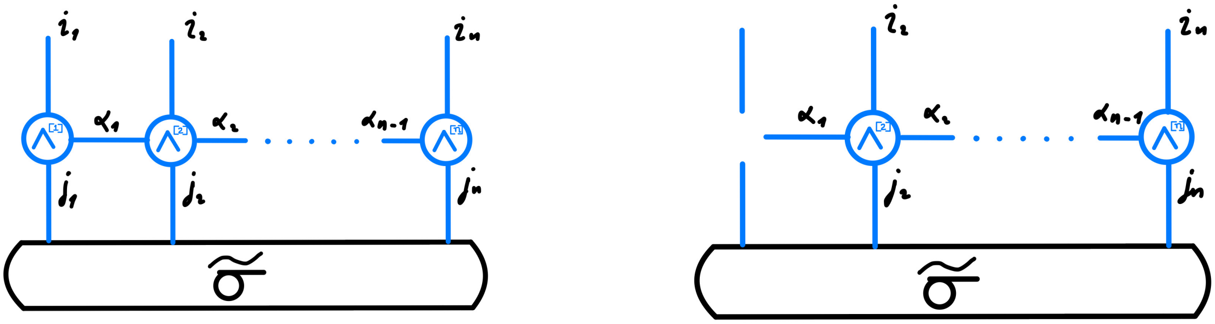


Figure 3.4: (left) Contraction of the MPO with the reshaped input configuration $\tilde{\sigma}$ i.e., $\chi_{i_1,i_2 \ldots i_n}$; (right) derrivative of $\chi_{i_1,i_2 \ldots i_n}$ w.r.t $\Lambda^{[1]}_{i_1,j_1,\alpha_1}$.

One of the very famour problems that plagues the gradient based optimization of DNN is the *vanishing gradient problem* or *barren plateau problem*. In a DNN the gradients are found through Back Propagation which basically uses the chain rule by multiplying the derrivatives from the outer layer to the inner initial layer to compute the derrivatives of the initial layer. This causes the gradient to decreases exponentially (really? and is it just the consquence of the backpropagation) with increasing number of layers. This means that the variational parameters of the network are not effectively updated and hence the model is not able to learn the wave function.

Although we only have a single layer of hidden units or in other words a single matrix of variational parameters, W, we still face the problem of *vanishing gradient* when we used the hybrid architecture. This is due to the fact that in an MPO we are contracting of n nodes which involves the multiplication over numbers.

However, the problem is only faced when we use complex variational parameters (why? and can we explain it. Probbly due to the different nature of the landscape which comes from the role of the complex cosh). We can solve this problem by initializing our network in such a way such that the second moment (variance) of the log gradients i.e., Var $\left[O_k^\theta(\mathbf{s})\right]$ is non-vanishing [6]. In this way we reverse engineer the overall variance of the gradient to a certain value of our choice. This can be interpretted as adding noise to our gradients which saves them from vanishing. In the follwoing we will calculate the Var $\left[O_k^\theta(\mathbf{s})\right]$.

Now as we can see calculating Var $\left[O_k^\theta(\mathbf{s})\right]$ from equation 3.6 is not so straightforward. We need to sum over $\chi_{i_1,i_2 \ldots i_n}$ together with $\frac{\partial\left(\chi_{i_1,i_2 \ldots i_n}\right)}{\partial \Lambda_{kp}^q}$. If we initialize our network parameters in a random gashion then using the follwoing property we can write.

$$\begin{aligned} \operatorname{Var}\left(\sum_{i=1}^n X_i\right) &= \sum_{i=1}^n \operatorname{Var}\left(X_i\right) + 2\sum_{i<j} \operatorname{Cov}\left(X_i, X_j\right) && (3.7) \\ &= \sum_{i=1}^n \operatorname{Var}\left(X_i\right) && (3.8) \end{aligned}$$

Where we ignored the second term about covariance because we are initiating our network in a random fashion.

Further for each $\chi_{i_1,i_2 \ldots i_n}$, we have contractions over different nodes which means that we have to calculate the variances of the multiplication of the random variables. For this we assume that our initialised network parameters have zero mean.

$$\begin{aligned} \operatorname{var}\left(X_1 \cdots X_n\right) &= E\left[\left(X_1 \cdots X_n\right)^2\right] - \left(E\left[X_1 \cdots X_n\right]\right)^2 && (3.9) \\ &= E\left[X_1^2 \cdots X_n^2\right] - \Big(E\left[\left(X_1\right] \cdots E\left[X_n\right]\right)^2 && (3.10) \\ &= E\left[X_1^2\right] \cdots E\left[X_n^2\right] - \left(E\left[X_1\right]\right)^2 \cdots \left(E\left[X_n\right]\right)^2 && (3.11) \\ &= \prod_{i=1}^n \Big(\operatorname{var}\left(X_i\right) + \left(E\left[X_i\right]\right)^2\Big) - \prod_{i=1}^n \left(E\left[X_i\right]\right)^2 && (3.12) \\ &= \prod_{i=1}^n \big(\operatorname{var}\left(X_i\right)\big) && (3.13) \end{aligned}$$

where we have used the fact that $E\left[X_i\right] = 0$.

Finally to make the calculation simpler we will approximate $\tan(\chi) \sim \chi$ for small χ. By using the above properties and inspecting the contractions in figure 3.4 we arrive at the follwing formula.

$$V_k = \begin{cases} \frac{1}{n_{jk}} \left(\frac{\prod_{m=1}^n n_{i_m} v}{n_k^{2n-1} n_D}\right)^{1/(2n-1)} & ;\text{ If } k = 1 \text{ or } k = n \\ \frac{1}{n_{jk}} \left(\frac{\prod_{m=1}^n n_{i_m} v}{n_k^{2n-1} n_0^{2n}}\right)^{1/2n-1)} & ;\text{ otherwise} \end{cases} \qquad (3.14)$$

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. $n_{i_1}, n_{i_2}, \ldots, n_{i_n}$ are the dimensions of the MPO legs labelled with $i_1, i_2, \ldots i_n$ and similarly $n_{j_1}, n_{j_2}, \ldots, n_{j_n}$ are the dimensions of the MPO legs labelled with $j_1, j_2, \ldots j_n$. n_D 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 vnishing gradient in the very beginning. 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.

$$\mathrm{var}\left[Z\right] = \mathrm{var}\left[\mathrm{Re}\left(Z\right)\right] + \mathrm{var}\left[\mathrm{Im}\left(Z\right)\right] \tag{3.15}$$

Equation 3.14 is an important result when it comes to creating hybrid architectures from TN and DNN. Since these kind of architectures are finding more and more applications, this result will serve as a good approximation to avoind getting stuck in the very beginning of the optimization

- Why does we face this problem for complex parameters in general
- 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

To do

- Include regularization schemes

Chapter 4

Results

4.1 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

$$\hat{H} = -\sum_{l=0}^{L-2} \hat{\sigma}_l^z \hat{\sigma}_{l+1}^z - g \sum_{l=0}^{L-1} \hat{\sigma}_l^x \tag{4.1}$$

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 4.1 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}.

4.1.1 16 spins data

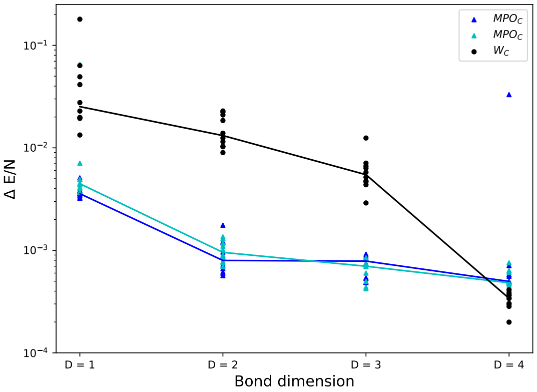


Figure 4.1: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 16 spins TFIM with OBC. The MPO decomposition used is [2, 8] for the cyan and [8, 2] for the blue color.

4.1.2 32 spin data

Below we plot some data with 32 spins

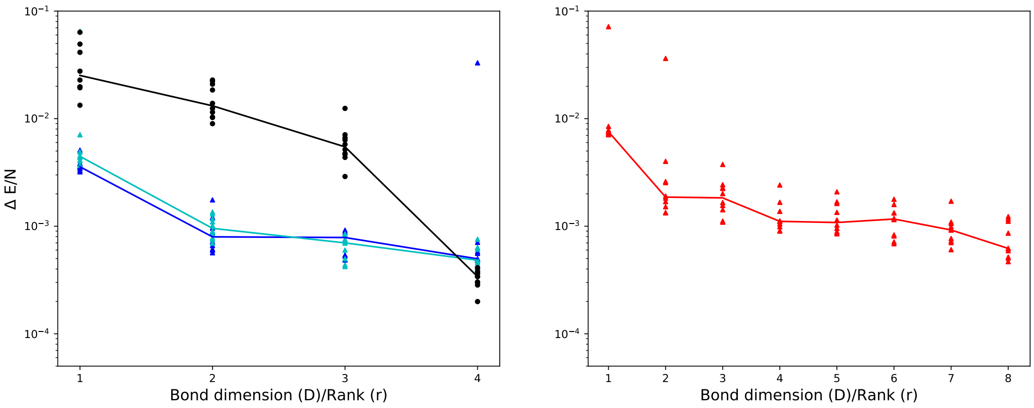


Figure 4.2: Left: [8, 2] decomposition. Right: [4, 2, 2] decomposition with $D_1 \leq 8$ and $D_2 = 2$

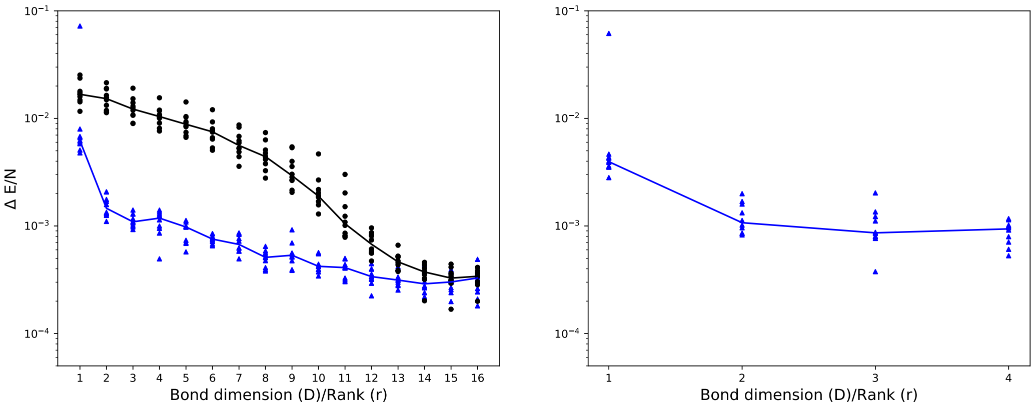


Figure 4.3: Left: [4, 4] decomposition. Right: [4, 2, 2] decomposition with $D_1 = 5$ and $D_2 \leq 4$

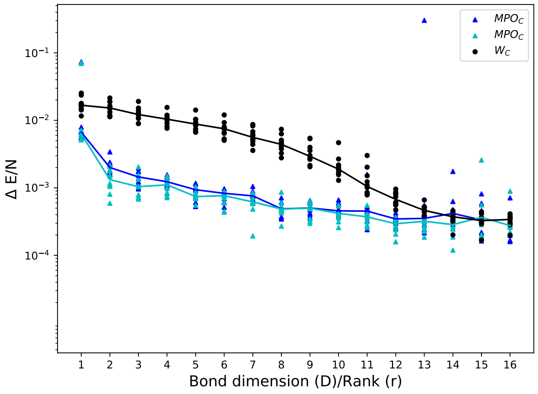
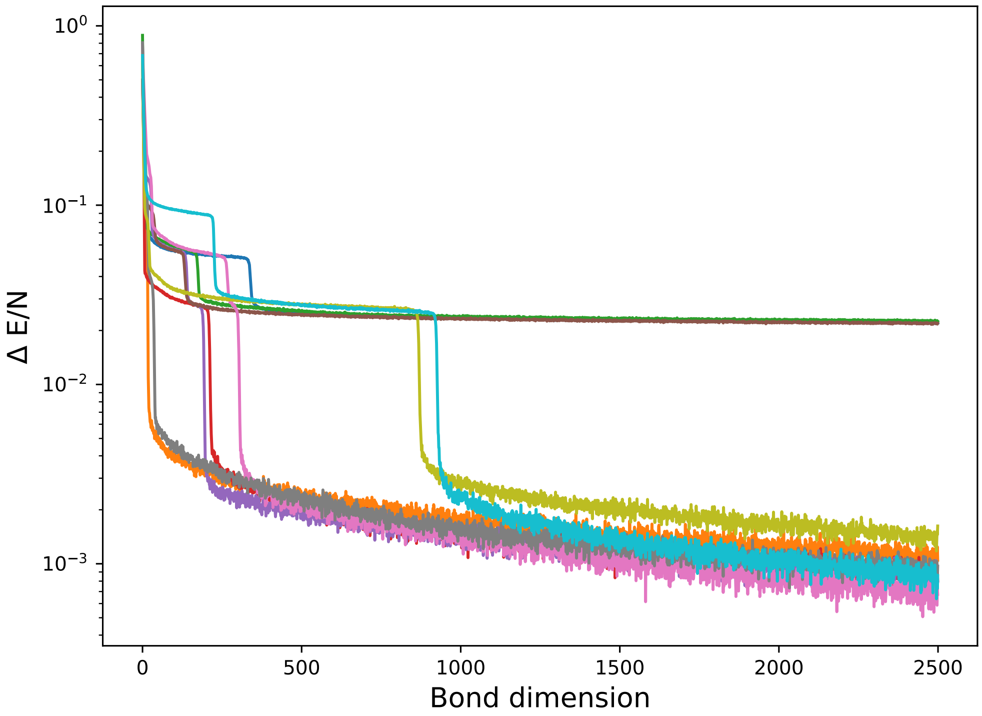


Figure 4.4: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 16 spins TFIM with OBC. The MPO decomposition used is [2,2, 4] for the cyan and [4, 2, 2] for the blue color.



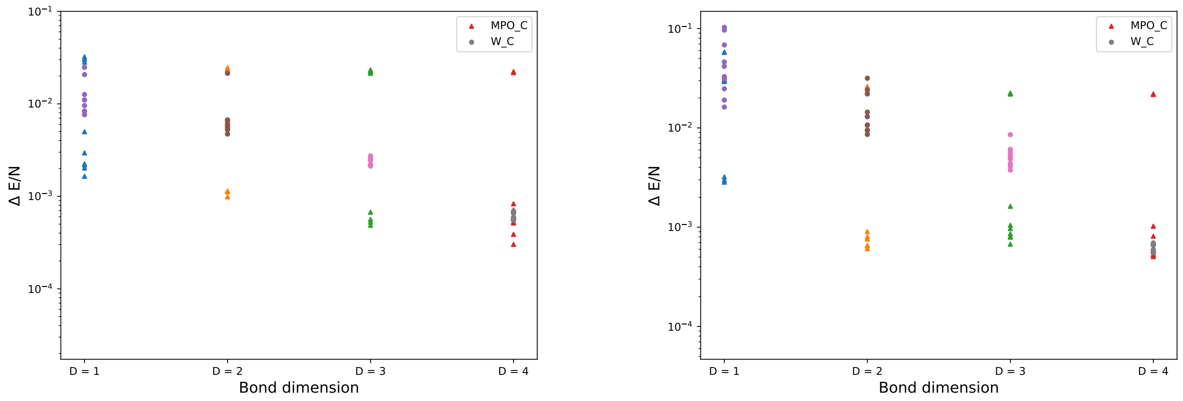


Figure 4.5: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. (Left) 2 node MPO with the decomposition [2, 16] in the input and the output dimensions of the legs. (Right) 2 node MPO with the decomposition [16, 2] in the input and the output dimensions of the legs

4.2 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.
- Compress plots along the x axis.

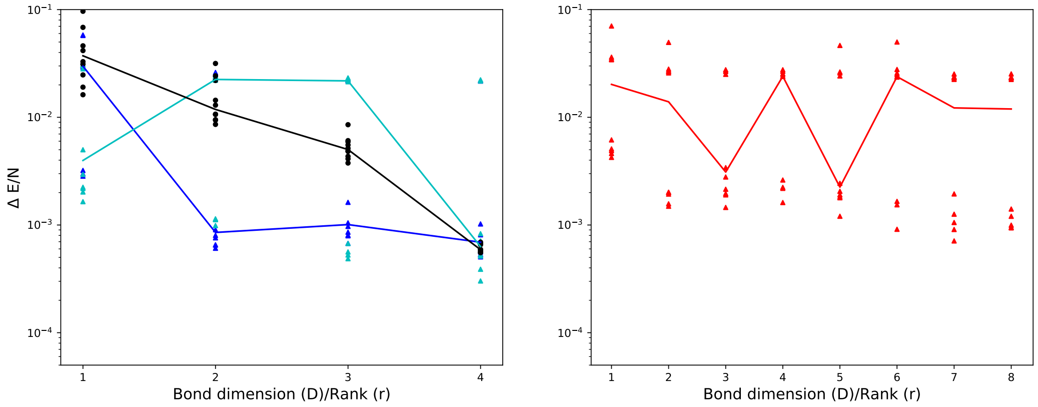


Figure 4.6: Left: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. The black blobs represent the data points for compressed W matrix via SVD, the cyan represent [2, 16] decomposition and the blue ones represent [16, 2] decomposition. Right: Decomposition of the blue cuve [16, 2] →[4, 2, 2] with $D_1 \leq 8$ and $D_2 = 2$

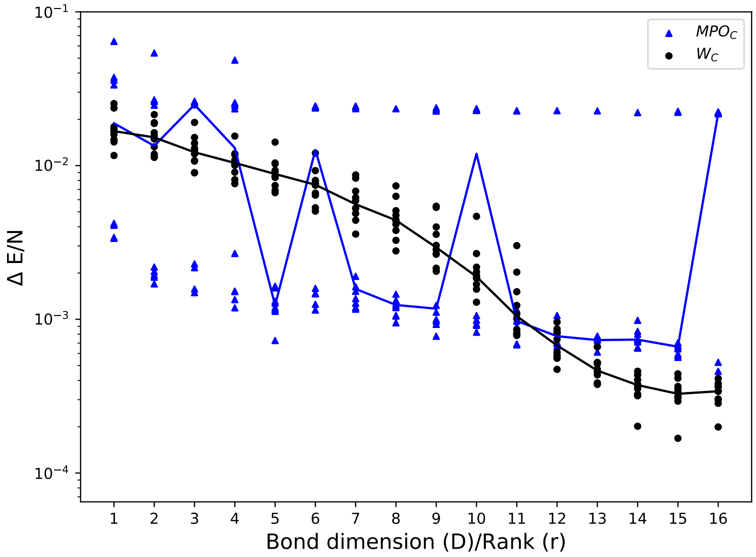


Figure 4.7: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. The black blobs represent the data points for compressed W matrix via SVD, the blue ones represent [4, 8] decomposition.could be a step size problem or just that [small, big] behaviour

- Plot the minima with a solid like and the median with a dashed line

4.3 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

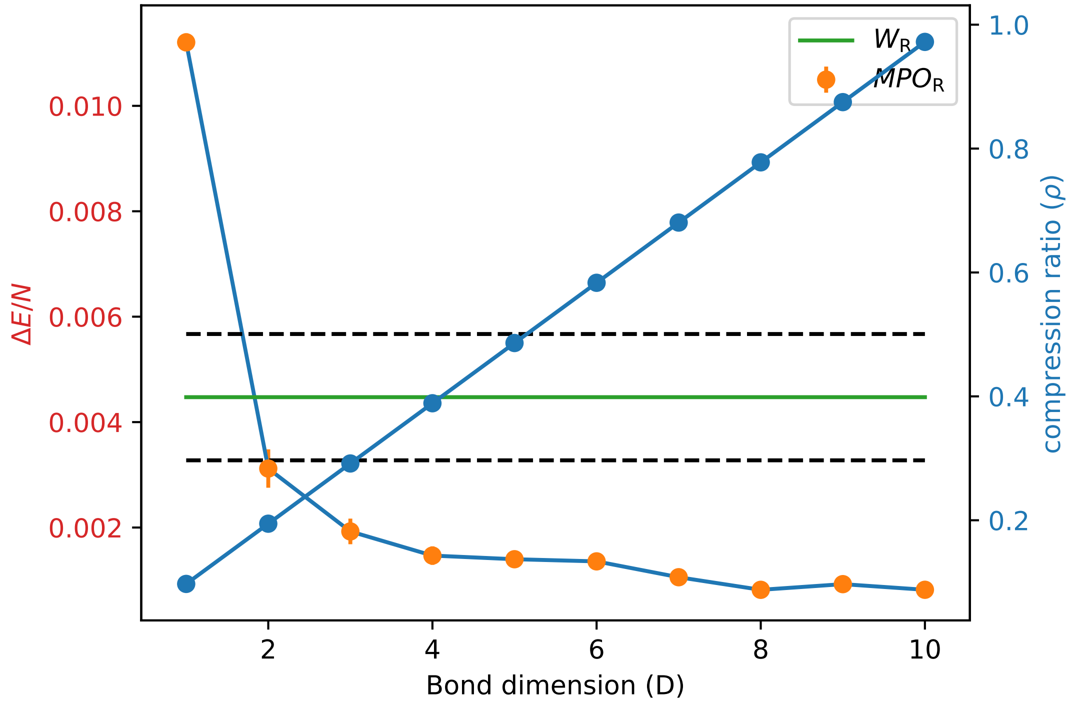


Figure 4.8: Plot for 12 spins with $\alpha = 3$ and [4, 3] decompotisiotn for Simple Gradient Descent

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 $D_2 = 4$ and $D_1 \leq 8$ done! unplotted yet
- 64 spins
  - simulate[2, 32] done! unplotted and the possible compression
  - simulate [32, 2]with $D \leq 4$, and the possible compression

Chapter 5

Conclusion and Outlook

5.1 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.

5.2 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 6

Appendix

6.0.1 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.

$$Var\left[O_k^\theta(\mathbf{s})\right] = Var\left[\partial_{\theta_k}\log\psi_\theta(\mathbf{s})\right] \tag{6.1}$$

$$\psi_\theta(\mathbf{s}) = e^{\sum_{i=1}^N a_i s_i} \prod_{i_1=1}^{L_1}\prod_{i_2=1}^{L_2} .... \prod_{i_n=1}^{M/L_1 L_2 ... L_{n-1}} 2\cosh\left(\chi_{i_1 i_2 .... i_n}\right) \tag{6.2}$$

$$\log\psi_\theta(\mathbf{s}) = \sum_{i=1}^N a_i s_i + \log(2) + \sum_{i_1=1, i_2=1 ...., i_n} \log\Big(\cosh\left(\chi_{i_1, i_2 ... i_n}\right)\Big) \tag{6.3}$$

Assuming $a_i = 0$

$$\frac{\partial \log\psi_\theta(\mathbf{s})}{\partial A_{kp}^q} = \sum_{i_1=1, i_2=1 ...., i_n} \left(\frac{\sinh\left(\chi_{i_1, i_2 ... i_n}\right)}{\cosh\left(\chi_{i_1, i_2 ... i_n}\right)}\right) \frac{\partial\left(\chi_{i_1, i_2 ... i_n}\right)}{\partial A_{kp}^q} \tag{6.4}$$

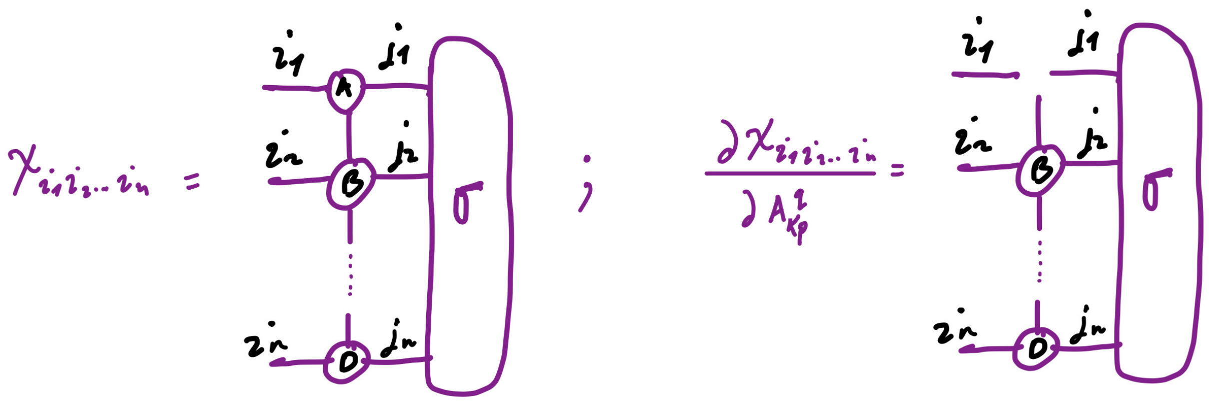


Figure 6.1: On the left we have the contraction of the reshaped input configuration with the MPO with ”n” number of nodes $\chi_{i_1,i_2,..,i_n}$and on the right we have a derrivative of this network with respect to the elements of a prticular node (here w.r.t the elements of the node A i.e., $\partial(\chi_{i_1,i_2...i_n})/\partial(A^q_{kp})$

.

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:

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

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.

$$Var\left[\frac{\partial \log \psi_\theta(\mathbf{s})}{\partial A^q_{kp}}\right] = \tag{6.5}$$

- 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.