

UNIVERSITÄT BONN

---

# Compressing Neural Quantum States using Matrix Product Operators

---

*Author:*

Junaid Akhter

January 31, 2022

*Supervisor:*

Professor Matteo Rizzi

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>3</b>
2.1	Many Body Problem . . . . .	3
2.2	Neural Quantum States (NQS) . . . . .	5
2.2.1	Deep Learning (DL) . . . . .	5
2.2.2	Representation of Many Body Wave function as Artificial Neural Neural Net-works . . . . .	7
2.2.3	Varational quantum montecarlo . . . . .	9
2.3	The curse of dimensionality . . . . .	11
<b>3</b>	<b>Compressing variational parameters inside NQS</b>	<b>12</b>
3.1	A brief introduction to Matrix Product Operators . . . . .	13
3.1.1	Interpretability . . . . .	14
3.2	Modified NQS ansatz . . . . .	14
3.2.1	Proper initialization of the network . . . . .	16
<b>4</b>	<b>Results</b>	<b>19</b>
4.1	Model . . . . .	19
4.1.1	16 spins data . . . . .	20
4.1.2	32 spin data . . . . .	20
4.1.3	64 spins data . . . . .	22
4.2	Comments on plots . . . . .	22
4.3	Pending decompositions . . . . .	24
<b>5</b>	<b>Conclusion and Outlook</b>	<b>26</b>
5.1	Conclusion . . . . .	26
5.2	Outlook . . . . .	26
<b>6</b>	<b>Appendix</b>	<b>27</b>
6.0.1	Appendix A.1: Proper gradient initialisation Already included in the main text	27



# Chapter 1

## Introduction

One of the main challenges that physicists are facing is solving the quantum many body problem. Quantum many body problem refers to the fact that the dimensionality 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 interpreted as a single layered feed forward neural network. To optimize the weights and biases (variational 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 efficiently 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 challenges that one faces in optimizing the variational parameters if one wants to simulate large 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 to solve 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:

$$|\Psi\rangle = \sum_{s_1, s_2, \dots, s_N=1}^p \psi(s_1, s_2, \dots, s_N) |s_1\rangle \otimes |s_2\rangle \otimes \dots \otimes |s_N\rangle \quad (2.1)$$

where the coefficients  $\psi(s_1, s_2, \dots, s_N)$  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  $\mathbf{s} = (s_1, s_2, \dots, s_N)$ . 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<sup>1</sup>. The direct consequence 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 important condensed matter phenomena beyond Landau's

---

<sup>1</sup>So far the best that has been done is to simulate around 43 spins 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 varied 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 antisymmetric 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 parameters. One of the oldest variational method is the *Hartree 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 systems 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 Carleo] 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 problems. They have been used to study both quantum spin systems without geometrical frustration [10 from [3]] and with geometrical frustration [15-21 in [3]], fermion systems [15, 24-29 in [3]] topological states [13, 31-36 in [3]], non-equilibrium states [5], excited states [21, 30, 37, 38], quantum systems with nonabelian or anyonic symmetries [38 in [3]]. Since the techniques in NQS based optimization are derived from Machine Learning, we are going to dive in a gentle introduction to Machine Learning and Deep Neural Networks (DNN) in the following sections.

### 2.2.1 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  $D = (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$  where  $\mathbf{x}_n \in \mathbb{R}^D$  is a vector and  $y_n \in \mathbb{R}$  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  $0 \rightarrow \text{dog}$  and  $1 \rightarrow \text{cat}$ ). We wish to estimate a predictor  $f(\cdot, \boldsymbol{\theta}) : \mathbb{R}^D \rightarrow \mathbb{R}$  where  $\boldsymbol{\theta}$  are the parameters of the predictor. The word *learning* in DL refers to finding the right set of parameters  $\boldsymbol{\theta}^*$  such that we fit the data well, i.e.,

$$f(\mathbf{x}_n, \boldsymbol{\theta}^*) \approx y_n \quad \text{for all } n = 1, \dots, N. \quad (2.2)$$

In this way our predictor has successfully learned the features of input data  $\mathbf{x}$  (cats and dogs). After learning is done, it should be able to categorize images of dogs and cats (different images than used during training) correctly with a good accuracy. For that purpose it is a usual practice to split the dataset into a training dataset  $D_{\text{train}}$  and a test dataset  $D_{\text{test}}$ . As is clear by their names, the network is trained on  $D_{\text{train}}$  and then its performance is tested on  $D_{\text{test}}$ . Following are the key components of performing a such a task using DL. In general following three ingredients are necessary to conduct 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*. The model used in Deep Learning is a Deep Neural Network (DNN) (hence the name Deep) which is discussed separately below.
- *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 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].

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  $\sigma_i(z)$  (figure 2.1).

$$a_i(\mathbf{x}) = \sigma_i(z^{(i)}) = \sigma_i(\mathbf{w}^{(i)} \cdot \mathbf{x} + b^{(i)}) \quad (2.3)$$

where  $\mathbf{w}^{(i)} = (w_1^{(i)}, w_2^{(i)}, \dots, w_d^{(i)})$  are the weights corresponding to the  $i_{th}$  neuron and  $b^{(i)}$  is the neuron-specific bias. Together the weights and biases,  $\mathbf{w}^{(i)} = (b^{(i)}, \mathbf{w}^{(i)})$  are the variational parameters in ML models. The non-linearities  $\sigma_i$  in a model can be chosen 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 2.1 right).

Once we have a model which produces an output corresponding to a certain input, the next thing to do is to use an 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 according to equation 2.4

$$\theta \rightarrow \theta' = \theta - \tau \frac{\partial l}{\partial \theta} \quad (2.4)$$

where  $\tau$  is a hyperparameters of the model known as the learning rate.

Equation 2.4 means that the variational parameters are updated in a manner so that at each iteration

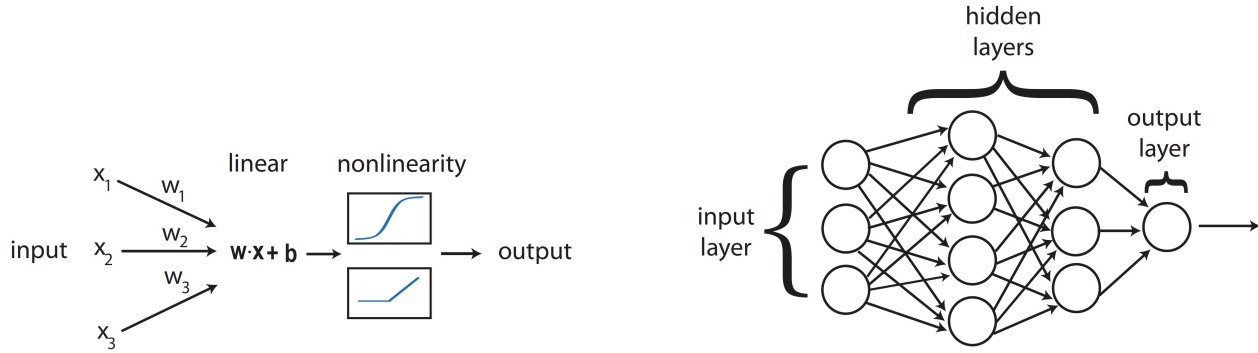


Figure 2.1: (left) The basis components of a neural network which consists of a linear transformation and followed by a non-linear transformation; (right) Neurons are stacked into layers with the output of one layer serving as the input to the next layer.[Reference from Marin]

we are moving towards the minimum of the cost function  $l$ . Intuitively one can think of the cost function as a landscape (figure ??) 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. **mention later in stochastic reconfiguration that the landscape has local minimas or saddle points which means that we need to use different variants of GD**

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., *convolutional neural networks* (CNNs), *Autoencoders*, *restricted Boltzmann machines* (RBM) etc.

## 2.2.2 Representation of Many Body Wave function as Artificial Neural Neural Networks

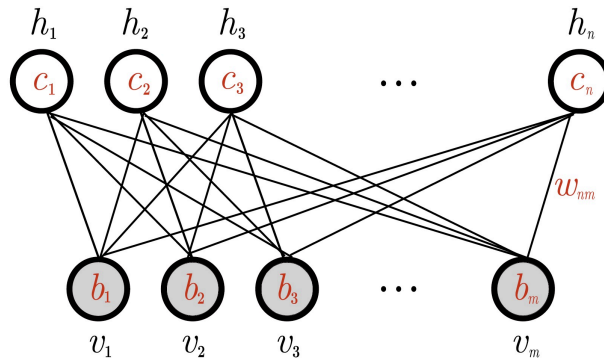


Figure 2.2: Interaction between hidden and visible units [notes]

Because DNN are universal function approximators they have also made their way to quantum

physics where they are used to represent the wave-function. In this work we are going to work with a special kind of Neural Network, restricted Boltzmann machine (RBM) as was also done in the original work [2].

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,  $\mathbf{h} = h_1, h_2, \dots, h_m$  and  $\mathbf{v} = v_1, v_2, \dots, v_n$  with the probability given by:

$$p(\mathbf{h}, \mathbf{v}) = e^{-E(\mathbf{h}, \mathbf{v})} \quad (2.5)$$

where

$$E(\mathbf{v}, \mathbf{h}) = - \sum_{i=1}^M \sum_{j=1}^N W_{ij} h_i v_j - \sum_{i=1}^M b_i h_i - \sum_{j=1}^N a_j v_j \quad (2.6)$$

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 equation 2.6. However, there is no intra-layer interaction within the visible or hidden layers (see figure 2.2). 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) and  $\theta$  with the variational parameters equation 2.5 is re-defined as the varational wave-function.

$$\psi_{\theta}(\mathbf{s}) = \sum_{\{h_i\}} e^{\sum_{ij} W_{ij} h_i s_j + \sum_i b_i h_i + \sum_j a_j s_j} \quad (2.7)$$

where  $s_i$  is the 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 equation 2.7 is a more compact form.

$$\psi_{\theta}(\mathbf{s}) = e^{\sum_i a_i s_i} \times \prod_{i=1}^M 2 \cosh[b_i + \sum_j W_{ij} s_j] \quad (2.8)$$

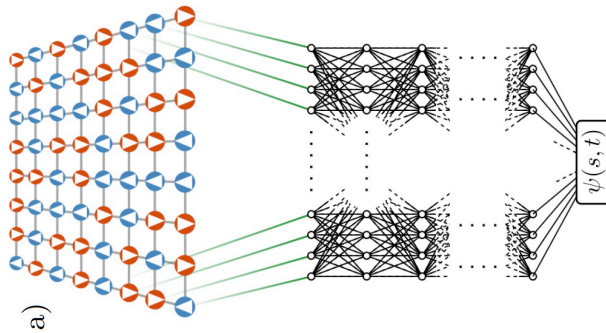


Figure 2.3: NQS representation of a many body wave-function

What equation 2.7 tells us is that for each input spin configuration  $\mathbf{s}$  and  $\theta$ , our network maps it to

a complex number  $\psi(\mathbf{s}, \boldsymbol{\theta})$  (see figure 2.3). This enables us to write the variational wave-function of a many body system in the following way.

$$|\Psi(\boldsymbol{\theta})\rangle = \sum_{\mathbf{s} \in \mathbb{Z}_p^N} \Psi(\mathbf{s}, \boldsymbol{\theta}) |\mathbf{s}\rangle \quad (2.9)$$

Equation 2.9 is used as an ansatz for the variational wave-function in VMC.

### 2.2.3 Variational quantum montecarlo

Variational Monte Carlo (VMC) [reference needed] method is a quantum MC method used to find ground states 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. 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 access to a variational wave-function from equation 2.8, the expectation value of a local operator  $\hat{O}$  can be calculated as follows:

$$\frac{\langle \psi(\boldsymbol{\theta}) | \hat{O} | \psi(\boldsymbol{\theta}) \rangle}{\langle \psi(\boldsymbol{\theta}) | \psi(\boldsymbol{\theta}) \rangle} = \sum_{\mathbf{s}, \mathbf{s}'} \frac{\psi_{\boldsymbol{\theta}}(\mathbf{s})^* \psi_{\boldsymbol{\theta}}(\mathbf{s}')}{\langle \psi(\boldsymbol{\theta}) | \psi(\boldsymbol{\theta}) \rangle} \langle \mathbf{s} | \hat{O} | \mathbf{s}' \rangle \quad (2.10)$$

$$= \sum_{\mathbf{s}} \frac{|\psi_{\boldsymbol{\theta}}(\mathbf{s})|^2}{\langle \psi(\boldsymbol{\theta}) | \psi(\boldsymbol{\theta}) \rangle} \sum_{\mathbf{s}'} \langle \mathbf{s} | \hat{O} | \mathbf{s}' \rangle \frac{\psi_{\boldsymbol{\theta}}(\mathbf{s}')}{\psi_{\boldsymbol{\theta}}(\mathbf{s})} \quad (2.11)$$

$$(2.12)$$

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

$$\frac{\langle \psi(\boldsymbol{\theta}) | \hat{O} | \psi(\boldsymbol{\theta}) \rangle}{\langle \psi(\boldsymbol{\theta}) | \psi(\boldsymbol{\theta}) \rangle} = \sum_{\mathbf{s}} p_{\boldsymbol{\theta}}(\mathbf{s}) O_{\text{loc}}^{\boldsymbol{\theta}}(\mathbf{s}) \equiv \langle \langle O_{\text{loc}}^{\boldsymbol{\theta}} \rangle \rangle_{\boldsymbol{\theta}} \quad (2.13)$$

where we introduced the local estimator

$$O_{\text{loc}}^{\boldsymbol{\theta}}(\mathbf{s}) = \sum_{\mathbf{s}'} \langle \mathbf{s} | \hat{O} | \mathbf{s}' \rangle \frac{\psi_{\boldsymbol{\theta}}(\mathbf{s}')}{\psi_{\boldsymbol{\theta}}(\mathbf{s})} \quad (2.14)$$

and the notation  $\langle \langle \cdot \rangle \rangle$  stands for an expectation value with respect to  $p_{\boldsymbol{\theta}}(\mathbf{s})$ . As we can see there are two sums in equation 2.12 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  $\sum_{\mathbf{s}'}$  can be evaluated efficiently. By contrast, the sum over all basis states,  $\sum_{\mathbf{s}}$  becomes infeasible because of the exponential growth of the Hilbert Space. At this point one has to resort to Monte Carlo (MC) sampling of  $p_{\boldsymbol{\theta}}(\mathbf{s})$  to efficiently estimate these expectation values. Using *Metropolis algorithm* (Explain in appendix maybe) we can obtain samples  $\{\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \dots, \mathbf{s}^{(N_{\text{MC}})}\}_{\mathbf{s} \sim p_{\boldsymbol{\theta}}(\mathbf{s})}$  and the expectation values can be approximated by the empirical mean as follows

$$\langle\langle O_{\text{loc}}^\theta \rangle\rangle_\theta \approx \frac{1}{N_{\text{MC}}} \sum_{n=1}^{N_{\text{MC}}} O_{\text{loc}}^\theta(\mathbf{s}^{(n)}) \quad (2.15)$$

- Definition of Variational monte carlo
- Metropolis Algorithm
- Calculating the expectation value of an operator using VMC
- for this purpose, only the functional form of  $\psi_\theta(\mathbf{s})$  must allow for efficient evaluation [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:

$$E(\theta) = \frac{\langle \Psi(\theta) | \hat{H} | \Psi(\theta) \rangle}{\langle \Psi(\theta) | \Psi(\theta) \rangle} \quad (2.16)$$

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. Nevertheless, 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  $\theta_k$  at each iteration of the optimization step  $k$ . This can be easily calculated from equation 2.16

$$\nabla_{\theta_k} E(\theta) = 2\text{Re} \left( \langle\langle (O_k^\theta)^* E_{\text{loc}}^\theta \rangle\rangle_\theta - \langle\langle (O_k^\theta)^* \rangle\rangle_\theta \langle\langle E_{\text{loc}}^\theta \rangle\rangle_\theta \right) \quad (2.17)$$

where  $E_{\text{loc}}^\theta(\mathbf{s}) = \sum_{\mathbf{s}'} \langle \mathbf{s} | \hat{H} | \mathbf{s}' \rangle \frac{\psi_\theta(\mathbf{s}')}{\psi_\theta(\mathbf{s})}$  is defined as in 2.14 is the local Energy. Further we also introduced the logarithmic derivatives  $O_k^\theta(\mathbf{s}) = \partial_{\theta_k} \log \psi_\theta(\mathbf{s})$  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.

$$\theta_k^{(n+1)} = \theta_k^n - \tau \partial_{\theta_k} E(\theta) |_{\theta=\theta^n} \quad (2.18)$$

It 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 get 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

$$\theta_k^{(n+1)} = \theta_k^n - \tau \mathcal{S}_{k,k'}^{-1} \partial_{\theta_{k'}} E(\theta) |_{\theta=\theta^n} \quad (2.19)$$

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. (2.19).

More about SR?

## 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 (larger systems or 2-D systems). Although  $N_{\text{var}}$  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 2.19 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.

- 

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

- Expectation value of Energy
- Optimization using Simple Gadiant 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 inside 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 reduction 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 introduction to Matrix Product Operators

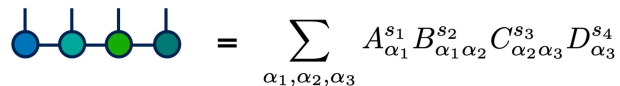
MPO are a class of *Tensor Networks (TN)* which were first introduced in physics to characterize the short-range entanglement in one-dimensional quantum many body systems [34-35 in [4]]. In mathematics it is known by the name tensor-train approximations [36-37 in [4]]. A TN is a factorization of a higher-order 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  $T_{i_1, i_2, \dots, i_n}$ . The number of values that a particular 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 linearly (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 diagrammatically.

For a detailed discussion on the diagrammatic notation 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 decomposition into smaller tensors (blobs). 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  $\chi$ . 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 losing 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 matrix  $W$  as a tensor-train. We see that using this representation we can gain compression in  $N_{\text{var}}$  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.



$$= \sum_{\alpha_1, \alpha_2, \alpha_3} A_{\alpha_1}^{s_1} B_{\alpha_1 \alpha_2}^{s_2} C_{\alpha_2 \alpha_3}^{s_3} D_{\alpha_3}^{s_4}$$

Figure 3.1: MPS



$$M_{s'_1 s'_2 s'_3 s'_4 s'_5 s'_6}^{s_1 s_2 s_3 s_4 s_5 s_6} = \sum_{\{\alpha\}} A_{s'_1}^{s_1 \alpha_1} A_{\alpha_1 s'_2}^{s_2 \alpha_2} A_{\alpha_2 s'_3}^{s_3 \alpha_3} A_{\alpha_3 s'_4}^{s_4 \alpha_4} A_{\alpha_4 s'_5}^{s_5 \alpha_5} A_{\alpha_5 s'_6}^{s_6 \alpha_6}$$

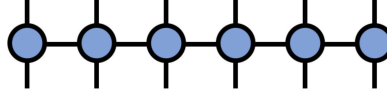


Figure 3.2: MPO

### 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 explanation.
- 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 time

## 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 dimension of the input legs of the MPO should be selected in such a way that the overall dimension is equal to the number of the size of the system/input configuration. 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 following formula:

$$N_{\text{var}} \sim D^2 \quad (3.1)$$

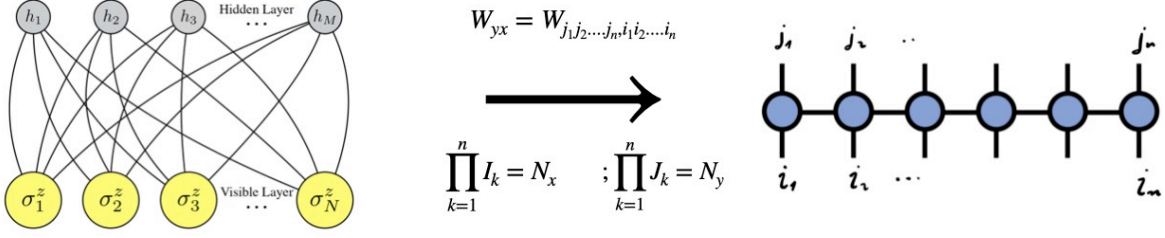


Figure 3.3: Replacement of the full matrix  $W$  with an MPO. The  $i$  and  $j$  indices need to be reversed. [create a better picture maybe]

The ansatz 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

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

where

$$\chi_{i_1 i_2 \dots i_n} = \sum_{\{\alpha\}} \sum_{j_1, j_2, \dots, j_n} \Lambda_{j_1, \alpha_1}^{[1], i_1} \Lambda_{j_2, \alpha_1 \alpha_2}^{[2], i_2} \dots \Lambda_{j_n, \alpha_{n-1}}^{[n], i_n} \sigma_{j_1, j_2 \dots j_n} \quad (3.3)$$

$\Lambda^i$  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 our network with this architecture and use our optimization scheme to get the ground state of the system. This is different than performing the SVD of the weight matrix at each iteration.

The logarithm of the wave-function needed to calculate the expectation values are given by

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

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 representation of the new ansatz
  - Constraint 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  $a_i = 0$

$$\frac{\partial \log \psi_\theta(\mathbf{s})}{\partial \Lambda_{kp}^q} = \sum_{i_1=1, i_2=1, \dots, i_n} \left( \frac{\sinh(\chi_{i_1, i_2 \dots i_n})}{\cosh(\chi_{i_1, i_2 \dots i_n})} \right) \frac{\partial(\chi_{i_1, i_2 \dots i_n})}{\partial \Lambda_{kp}^q} \quad (3.5)$$

$$= \sum_{i_1=1, i_2=1, \dots, i_n} \tanh(\chi_{i_1, i_2 \dots i_n}) \frac{\partial(\chi_{i_1, i_2 \dots i_n})}{\partial \Lambda_{kp}^q} \quad (3.6)$$

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

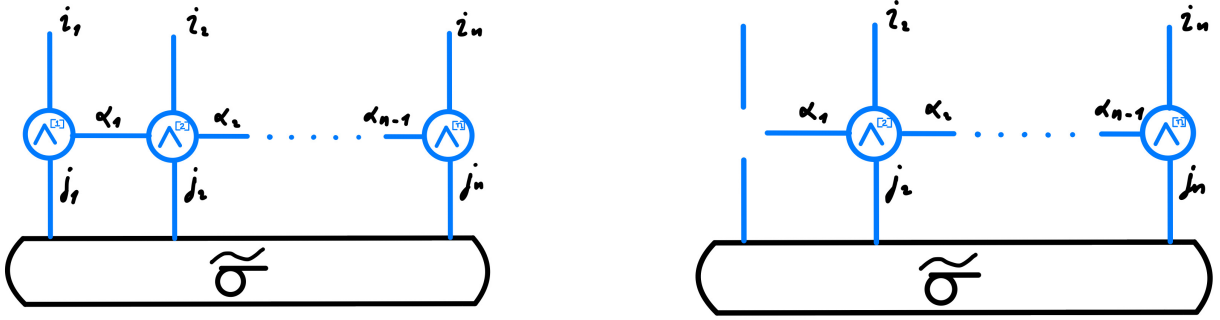


Figure 3.4: (left) Contraction of the MPO with the reshaped input configuration  $\tilde{\sigma}$  i.e.,  $\chi_{i_1, i_2 \dots i_n}$ ; (right) derivative of  $\chi_{i_1, i_2 \dots i_n}$  w.r.t  $\Lambda_{i_1, j_1, \alpha_1}^{[1]}$ .

One of the very famous 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 derivatives from the outer layer to the inner initial layer to compute the derivatives of the initial layer. This causes the gradient to decrease exponentially (really? and is it just the consequence 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. Probably 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.,  $\text{Var} [O_k^\theta(\mathbf{s})]$  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 interpreted as adding noise to our gradients which saves them from vanishing. In the following we will calculate the  $\text{Var} [O_k^\theta(\mathbf{s})]$ .

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

$$\text{Var} \left( \sum_{i=1}^n X_i \right) = \sum_{i=1}^n \text{Var} (X_i) + 2 \sum_{i < j} \text{Cov} (X_i, X_j) \quad (3.7)$$

$$= \sum_{i=1}^n \text{Var} (X_i) \quad (3.8)$$

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

Further for each  $\chi_{i_1, i_2 \dots i_n}$ , 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.

$$\text{var} (X_1 \cdots X_n) = E [(X_1 \cdots X_n)^2] - (E [X_1 \cdots X_n])^2 \quad (3.9)$$

$$= E [X_1^2 \cdots X_n^2] - (E [X_1] \cdots E [X_n])^2 \quad (3.10)$$

$$= E [X_1^2] \cdots E [X_n^2] - (E [X_1])^2 \cdots (E [X_n])^2 \quad (3.11)$$

$$= \prod_{i=1}^n (\text{var} (X_i) + (E [X_i])^2) - \prod_{i=1}^n (E [X_i])^2 \quad (3.12)$$

$$= \prod_{i=1}^n (\text{var} (X_i)) \quad (3.13)$$

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

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

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

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  $v$ .  $n_{i_1}, n_{i_2}, \dots, n_{i_n}$  are the dimensions of the MPO legs labelled with  $i_1, i_2, \dots, i_n$  and similarly  $n_{j_1}, n_{j_2}, \dots, n_{j_n}$  are the dimensions of the MPO legs labelled with  $j_1, j_2, \dots, j_n$ .  $n_D$  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 vanishing gradient in the very beginning. However, in a computer we have to initialize the real and the complex parts of the separately and use the following property to get the variance of the complex random number.

$$\text{var}[Z] = \text{var}[\text{Re}(Z)] + \text{var}[\text{Im}(Z)] \quad (3.15)$$

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 avoid 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 derivative 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 Conditions (OBC), the hamiltonian of which is given as follows

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

where  $\sigma_l^z$  and  $\sigma_l^x$  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  $\epsilon_{\text{rel}} = (E_{\text{NQS}^{\text{new}}}(\alpha/D) - E_{\text{exact}}) / |E_{\text{exact}}|$ , 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 particular 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 result 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  $N_{\text{var}}$ . 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  $N_{\text{var}}$  without losing 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  $N_{\text{var}}$ .

#### 4.1.1 16 spins data

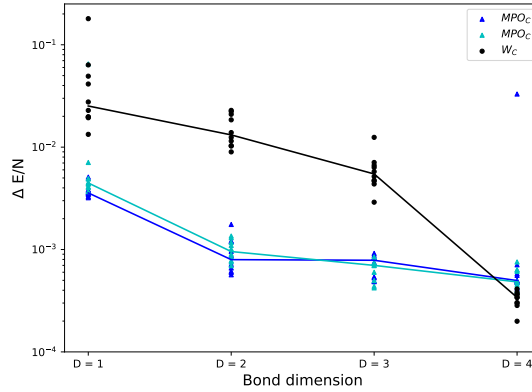


Figure 4.1: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 16 spins TFIM with OBC. The MPO decomposition used is  $[2, 8]$  for the cyan and  $[8, 2]$  for the blue color.

#### 4.1.2 32 spin data

Below we plot some data with 32 spins

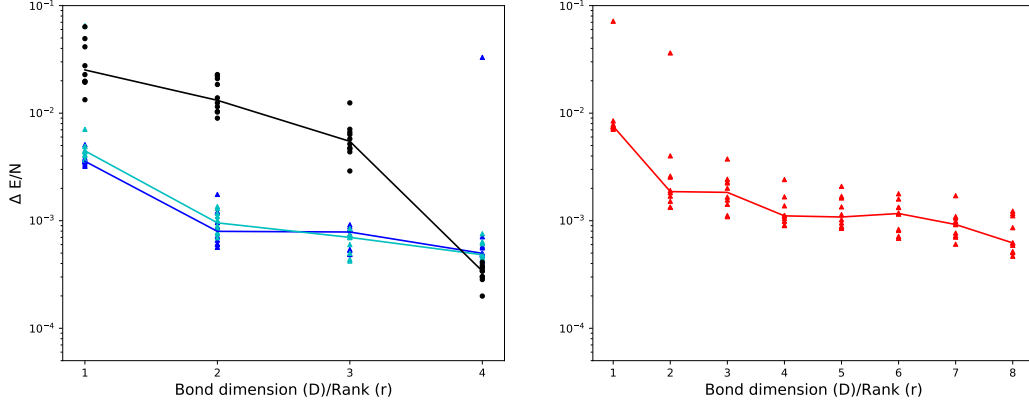


Figure 4.2: Left:  $[8, 2]$  decomposition. Right:  $[4, 2, 2]$  decomposition with  $D_1 \leq 8$  and  $D_2 = 2$

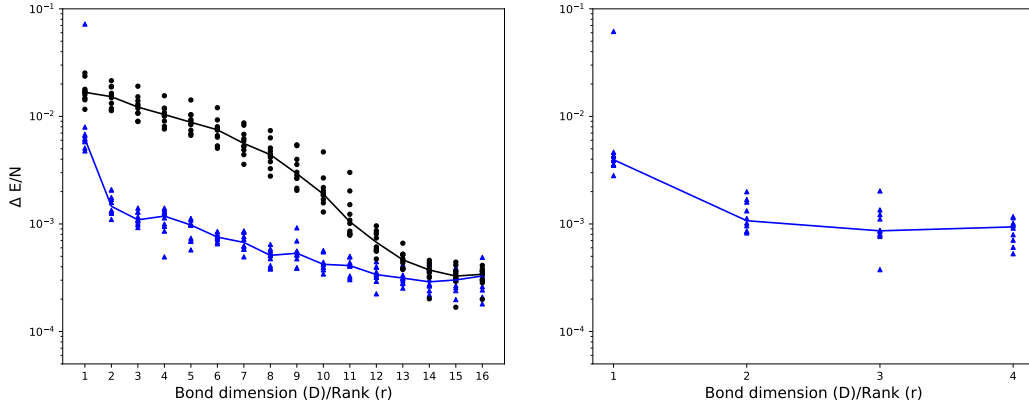


Figure 4.3: Left:  $[4, 4]$  decomposition. Right:  $[4, 2, 2]$  decomposition with  $D_1 = 5$  and  $D_2 \leq 4$

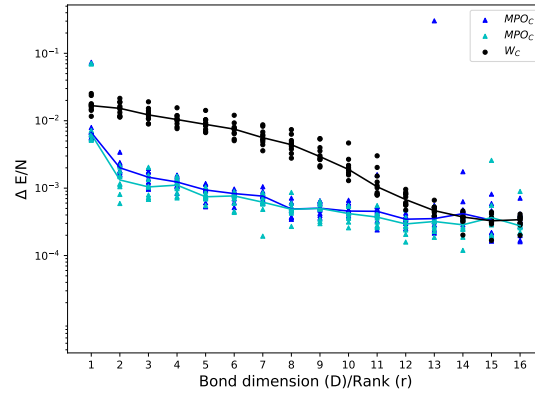


Figure 4.4: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 16 spins TFIM with OBC. The MPO decomposition used is  $[2, 2, 4]$  for the cyan and  $[4, 2, 2]$  for the blue color.



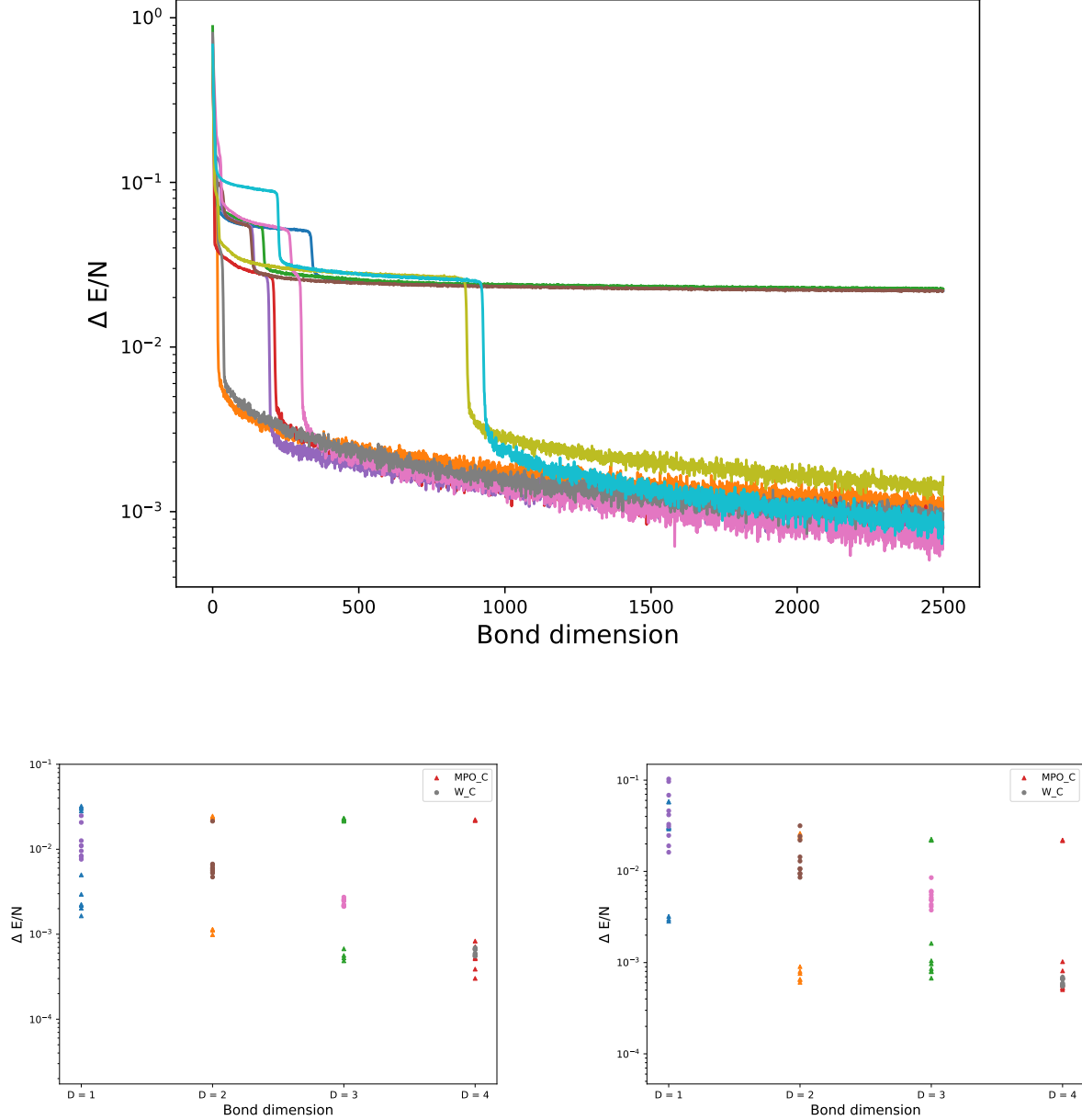


Figure 4.5: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. (Left) 2 node MPO with the decomposition  $[2, 16]$  in the input and the output dimensions of the legs. (Right) 2 node MPO with the decomposition  $[16, 2]$  in the input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

### 4.1.3 64 spins data

## 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.

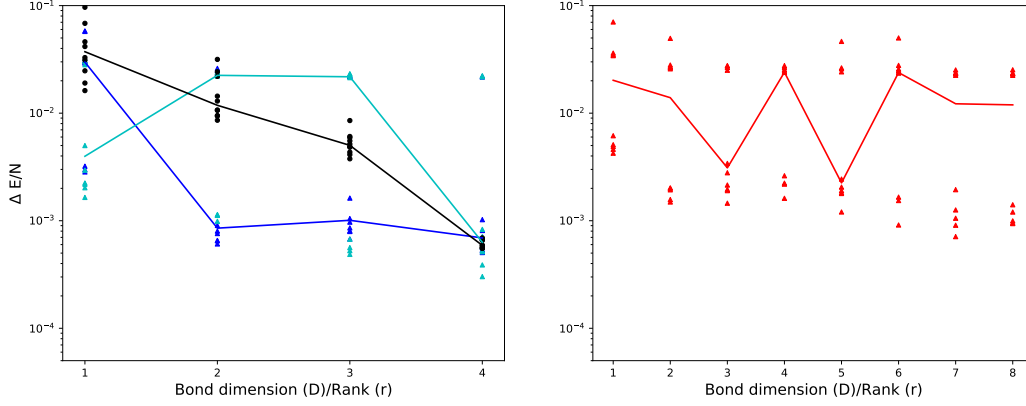


Figure 4.6: Left: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. The black blobs represent the data points for compressed  $W$  matrix via SVD, the cyan represent [2, 16] decomposition and the blue ones represent [16, 2] decomposition.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

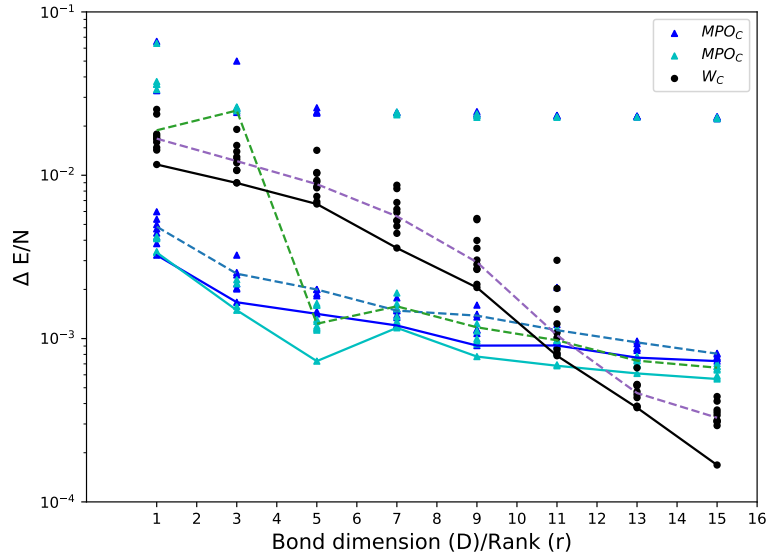


Figure 4.7: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 32 spins TFIM with OBC. The black blobs represent the data points for compressed  $W$  matrix via SVD, the blue ones represent [8, 4] and the cyan ones represent [4, 8] decomposition. could be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

- Compress plots along the x axis.
- Plot the minima with a solid line 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

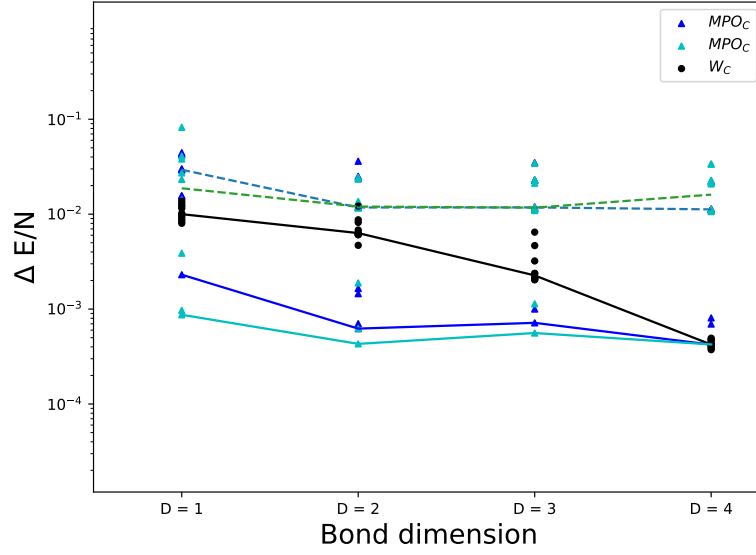


Figure 4.8: Comparison of performance of the architecture with an MPO and with the truncated full matrix for 64 spins TFIM with OBC. The black blobs represent the data points for compressed  $W$  matrix via SVD, the blue ones represent  $[32, 2]$  and the cyan ones represent the  $[2, 32]$  decomposition.

### 4.3 Pending decompositions

- red → Finished, green → Running, blue → To be run
- 16 spins
  - compress  $[4, 4] \rightarrow [2, 2, 4]$  with  $D_1 \leq 5$ , and  $D_2 = 4$  Done! unplotted (exp. to be same as  $[4, 4, 2]$ )
  - compress  $[8, 2] \rightarrow [4, 2, 2]$  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  $[16, 2] \rightarrow [8, 2, 2]$  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

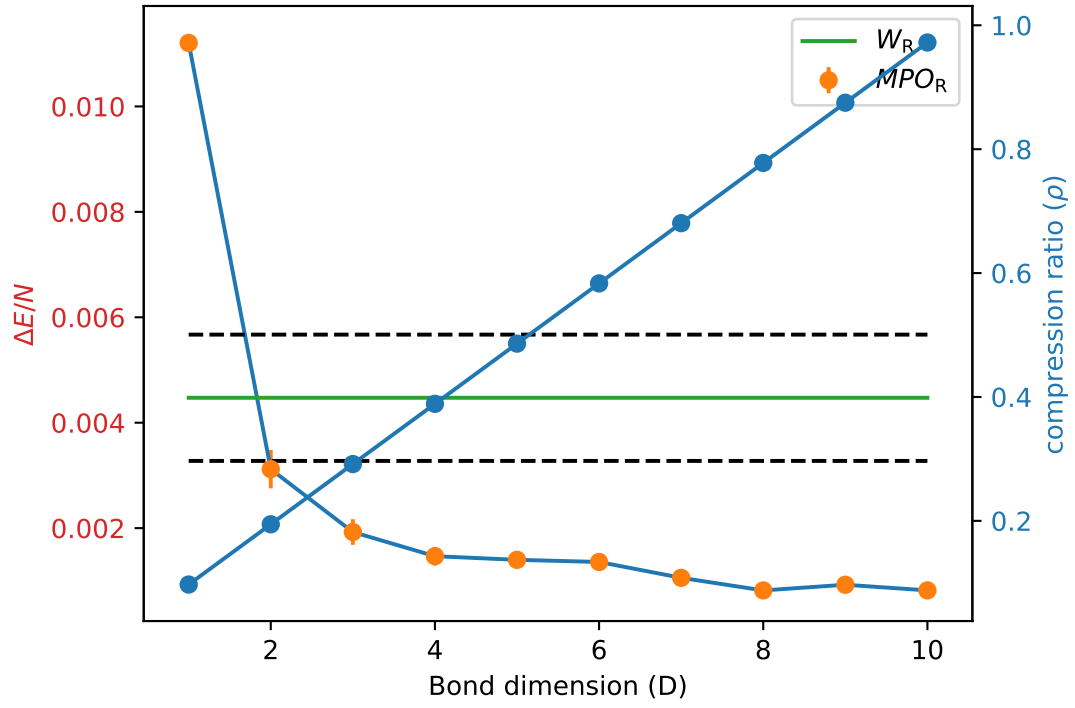


Figure 4.9: Plot for 12 spins with  $\alpha = 3$  and  $[4, 3]$  decomposition for Simple Gradient Descent

## 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 whether 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 information 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 the problem 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 wavefunction 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 in the energy landscape.

While we did not face such problem when dealing with the real parameters but we did face such issues when we started incorporating the MPO inside 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 makes sure that we have some fluctuations 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.

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

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

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

Assuming  $a_i = 0$

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

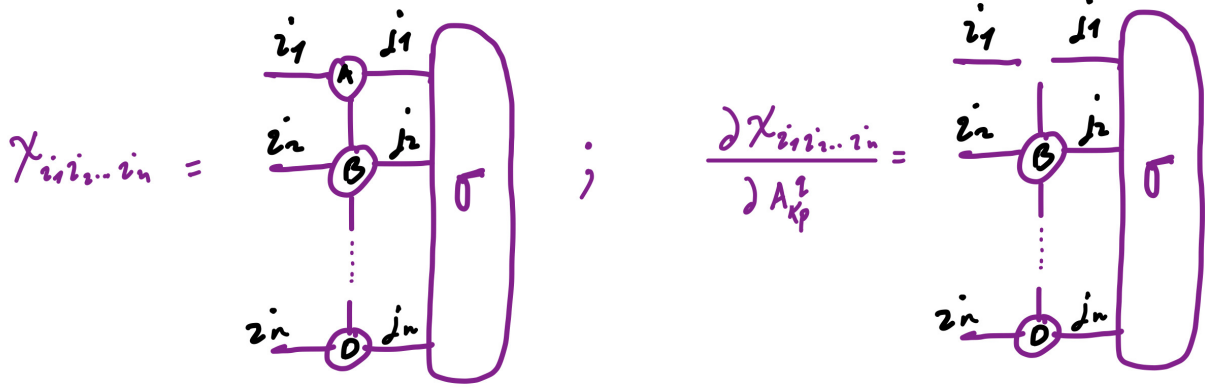


Figure 6.1: On the left we have the contraction of the reshaped input configuration with the MPO with "n" number of nodes  $\chi_{i_1, i_2, \dots, i_n}$  and on the right we have a derivative of this network with respect to the elements of a particular node (here w.r.t the elements of the node A i.e.,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

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 approximation for the  $\tan(\chi) \simeq \chi$ . Using these approximations we can get to the following result:

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

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  $v$ .  $n_{i_1}, n_{i_2}, \dots, n_{i_n}$  are the dimensions of the MPO legs labelled with  $i_1, i_2, \dots, i_n$  and similarly  $n_{j_1}, n_{j_2}, \dots, n_{j_n}$  are the dimensions of the MPO legs labelled with  $j_1, j_2, \dots, j_n$ .  $n_D$  is the bond dimension of the MPO which we have assumed to be the same throughout.

$$Var \left[ \frac{\partial \log \psi_{\theta}(\mathbf{s})}{\partial A_{kp}^q} \right] = \quad (6.5)$$

- 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.